Intermittent Computing with Dynamic Voltage and Frequency Scaling

Saad Ahmed
SBA School of Science and Engineering, LUMS University
Pakistan
saad.ahmed@lums.edu.pk

Qurat ul Ain
SBA School of Science and Engineering, LUMS University
Pakistan
18030035@lums.edu.pk

Junaid Haroon Siddiqui
SBA School of Science and Engineering, LUMS University
Pakistan
junaid.siddiqui@lums.edu.pk

Luca Mottola
Politecnico di Milano, Italy and RI.SE Sweden and Uppsala University, Sweden
luca.mottola@polimi.it

Muhammad Hamad Alizai
SBA School of Science and Engineering, LUMS University
Pakistan
hamad.alizai@lums.edu.pk

Abstract

We present D^{2}VFS, a run-time technique to intelligently regulate supply voltage and accordingly reconfigure clock frequency of intermittently-computing devices. These devices rely on energy harvesting to power their operation and on small capacitors as energy buffer. Statically setting their clock frequency fails to achieve energy efficiency, as the setting remains oblivious of fluctuations in capacitor voltage and of their impact on a microcontroller operating range. In contrast, D^{2}VFS captures these dynamics and places the microcontroller in the most efficient configuration by regulating the microcontroller supply voltage and changing its clock frequency. Our evaluation shows that D^{2}VFS markedly increases energy efficiency; for example, ultimately enabling a 30-300% reduction of workload completion times.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems

General Terms

Design, Measurement, Experimentation

Keywords

Transiently Powered Computers, Intermittent Computing, Dynamic Voltage and Frequency Scaling

1 Introduction

Intermittently-computing devices operate using energy harvested from their environment. This form of energy provisioning is generally highly variable across space and time [11]. To ameliorate the resulting fluctuations in energy supply, devices collect energy into ephemeral storage, typically a capacitor. As long as the capacitor voltage is below a predetermined power-up threshold, the device rests dormant until the buffered energy is sufficient to boot.

An *active epoch* then starts when the device operates until power fails and the device shuts down. A phase of solely charging then resumes until the device can boot again. These charge-discharge cycles are frequent, as miniaturization required to realize medical implants or visions of "smart dust" prompts energy storage facilities to be minimized [11]. Specialized software support is required to sustain computations in such intermittent settings, for example, using checkpoints [40, 6, 12] or programming abstractions with transactional semantics [31, 15, 33].

Opportunity. Energy efficiency of intermittently-computing devices depends on both the current capacitor voltage and processor speed [1]. Compared to battery-powered devices, capacitor voltage drops way more rapidly as the device extracts energy from it. Microcontroller units (MCUs) configured with static clock frequency fail to react to these changes, compromising energy efficiency.

As shown in Fig. 1, higher frequencies are more energy efficient, but limit the MCU operation in a narrow interval of the microcontroller's supply voltage values. Differently, lower frequencies consume more energy due to a longer duration of clock cycles, but enable a larger range of supply voltage values. For instance, the popular MSP430-series MCUs run with supply voltages as low as 1.8 V at 1 MHz, but are unable to run any lower than 3.3 V at 16 MHz.

Nonetheless, MCUs using higher clock frequencies tend to operate in short bursts, inducing higher overhead needed to sustain computations across power cycles, for example, when checkpointing the application state on non-volatile memory [40]. Configuring MCUs with lower clock frequencies does not solve the problem either, as fewer cycles are executed per unit of energy, besides reducing processing speed.

International Conference on Embedded Wireless Systems and Networks (EWSN) 2020
17–19 February, Lyon, France © 2020 Copyright is held by the authors.
Permission is granted for indexing in the ACM Digital Library
ISBN: 978-0-9949886-4-5

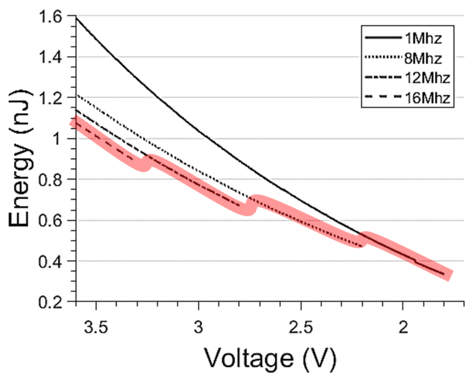


Figure 1: Variations in energy consumption per clock cycle with varying supply voltages. *The graph plots the energy consumption for the most commonly used and factory calibrated frequencies of MSP430G2553 MCUs. Higher frequencies are energy efficient but only operate in a limited range of supply voltage. Lower frequencies consumer more energy but enjoy a larger supply voltage range. The highlighted region indicates the best performing clock setting in a certain voltage range.*

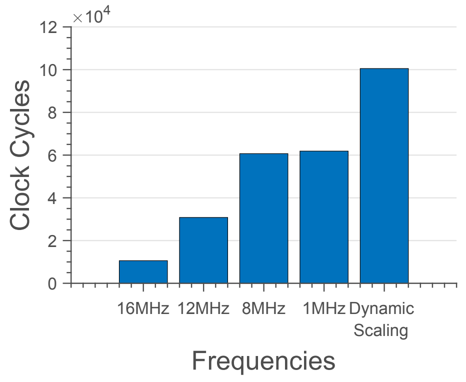


Figure 2: First order approximation of clock cycles for static configurations (1 MHz, 8 MHz, 12 MHz, and 16 MHz) and dynamic frequency scaling within a single charge of the capacitor (10 μF). *Dynamic scaling offers a substantial margin for improvement. The plot does not account for the overhead to enable dynamic scaling.*

D^{2}VFS[1] is a voltage and frequency scaling technique that seeks to take advantage from these dynamics, as we further discuss in Sec. 2. In Fig. 1, D^{2}VFS dynamically reconfigures the system to make it proceed through the highlighted route of clock frequencies, harnessing the maximum range of supply voltage values, and yet selecting the highest possible frequency in a given range. This effectively makes the system operate in the most efficient configuration given a certain capacitor voltage. Our approach gradually de-accelerates the MCU to maximize the number of available clock cycles, and hence the computations performed in an active epoch.

We implement D^{2}VFS through a hardware-software codesign, as described in Sec. 3. The hardware part includes voltage detectors to capture the current capacitor voltage and a voltage regulator to undervolt the device. The software part reacts to the voltage detectors providing different signals to reconfigure the clock frequency as soon as the capacitor voltage crosses marked boundaries.

Benefits and road-map. The dynamic selection of clock speed and undervolting of MCU in D^{2}VFS increases the number of clock cycles available in an active epoch by 40-900% compared to different static frequency configurations. This bears beneficial cascading effects on other key performance metrics.

D^{2}VFS reduces the peak energy demand, enabling a reduction of *up to one-sixth* in the size of the energy buffers necessary for completing a given workload, cutting charging times and enabling smaller device footprints. This is crucial in applications such as wearables [16] and biomedical implants [5]. Furthermore, D^{2}VFS consumes fewer checkpoints to complete a workload due to increasing availability clock cycles. Sparing checkpoints lets the system progress farther on a single charge, cutting down the time to complete a workload up to 300%.

Our quantitative assessment in this respect is two-pronged. Sec. 4 reports on the performance of D^{2}VFS based on three benchmarks across two existing system support (i.e., Hibernus [6] and MementOS [40]) and synthetic power profiles that allow fine-grained control on executions and accurate interpretation of results. Sec. 5 investigates the impact of D^{2}VFS using power traces obtained from highly diverse harvesting sources. We show that the results hold across these different power traces, demonstrating the general applicability of D^{2}VFS and its performance impact.

We end the paper with a brief survey of related approaches in Sec. 6 and with concluding remarks in Sec. 7.

2 Overview

We first present relevant background on dynamic voltage and frequency scaling (DVFS). We then scrutinize the potential benefits of employing DVFS in intermittently-computing devices, before highlighting the key challenges in preserving those benefits in a practical implementation.

Background. DVFS is a combination of two power saving techniques, i.e., voltage scaling and frequency scaling, originally meant to conserve battery in mobile devices. It configures the processor to work in different *operational zones* that are defined by a tuple consisting of a frequency f and a voltage V ($\{f,V\}$). This type of power saving is different from standby or hibernate power states, as it allows devices to continue performing tasks with a reduced amount of power. The technology is used in almost all modern computer hardware to improve battery life while still maintaining ready-to-compute performance.

The power dissipated per unit of time by a processor chip is $P = C \cdot V^2 \cdot A \cdot f$, where C is the capacitance being switched per clock cycle, V is voltage, A is the activity factor indicating the average number of switching events undergone by transistors in the chip, and f is the switching frequency. Voltage therefore dominates power. The voltage required for

[1]Discrete Dynamic Voltage and Frequency Scaling

stable operation is, however, determined by the frequency at which the processor is clocked and can be reduced if the frequency is also reduced. Modern computers allow software control of supply voltages and frequency, but embedded processors may require hardware modifications, such as voltage regulators, to do so.

Scrutinizing DVFS. The benefits of DVFS are well understood in mainstream computing. Intuitively, it also appears to be an attractive proposition for intermittently-computing devices, where capacitor voltage uncontrollably straddles across a microcontroller's operational range.

As a first order approximation of the benefits of DVFS in this domain, using existing models [1] we calculate the number of clock cycles accumulated in a single active epoch for the dynamic frequency route highlighted in Fig. 1, but without accounting for additional overhead required to enable this functionality. Fig. 2 depicts the results, which are encouraging compared to the most common static frequency configurations for the MSP430 platform. Ideally, the increase in the number of available MCU cycles ranges from 60% to *one order of magnitude*, as we stretch our comparison from the lowest (1 MHz) to the highest (16 MHz) configurable frequency. These approximate results support our hypothesis that DVFS for intermittently-computing devices is indeed worth a try.

Challenges. Employing DVFS in intermittently-computing devices is, though, not straightforward. MCUs employed for intermittently-computing platforms often lack hardware support for monitoring and controlling supply voltage.

Energy constraints are hard, imposing firm restrictions on the power dissipated by any additional hardware to be deployed to that end, such as voltage detectors and regulators. The former are needed to detect when the voltage crosses marked boundaries that could trigger frequency scaling, whereas the latter are necessary for undervolting the MCU at the minimum required voltage given the operating frequency. Nonetheless, the narrower the distance between these voltage boundaries, the more fine-grained frequency scaling can become, but the higher is the overhead of voltage detection and regulation.

Thus, we are to identify a sweet spot in this cost-benefit spectrum. In addition, frequency scaling incurs processing overhead, as the software must read the frequency configurations provided by MCU manufacturers from some form of flash memory and update appropriate clock registers. This overhead needs to be minimized as well.

3 D^{2}VFS

The principle operation of D^{2}VFS is similar to traditional DVFS techniques, i.e., to place the MCU in different operational zones, yet D^{2}VFS stands apart for three reasons:

1) unlike mainstream platforms, embedded MCUs typically lack on-chip DVFS engines to scale voltage and frequency on the fly;
2) energy in intermittently-computing devices is extremely scarce, imposing tight constraints on any additional hardware support;
3) switching between different operational zones is much more frequent and triggered by the varying capacitor voltage as it travels through the MCU operational range.

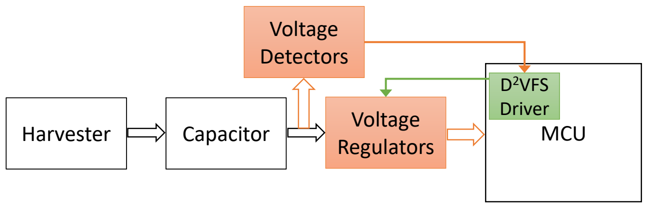


Figure 3: D^{2}VFS architecture. *Voltage detectors intercept the capacitor voltage to detect changepoints. The D^{2}VFS driver scales the frequency at these changepoints and reconfigures the voltage regulators to deliver the minimum required voltage for reducing energy consumption.*

We begin with describing the high level architecture of D^{2}VFS, followed by platform-specific implementation details. To make the discussion concrete, we target MSP430-class MCUs as arguably representative of intermittently-computing platforms for both academic [43, 18] and commercial purposes [8]. Nonetheless, our techniques apply more generally and have a foundational nature.

3.1 Architecture

Voltage and frequency scaling in D^{2}VFS is achieved through a hardware-software co-design.

Components. The hardware part is responsible for *detecting* discrete changepoints in capacitor voltage as well as for *regulating* the supply voltage to the MCU. The software part is responsible for *reconfiguring* the MCU clock frequency, the voltage detectors, and the voltage regulators.

The block diagram in Fig. 3 shows the main hardware and software components that enable D^{2}VFS for embedded MCUs. The capacitor voltage is intercepted by voltage detectors. When these detect a changepoint, an interrupt is fired whose service routine triggers the D^{2}VFS driver. This ascertains the current changepoint by scanning the state of all detectors and decides whether or not to scale the voltage and frequency at the current changepoint. If scaling is required, the D^{2}VFS driver reconfigures both the frequency and the output of the voltage regulators to place the MCU into a different operational zone.

Tradeoffs. The granularity of voltage detection and regulation is key to the efficiency of D^{2}VFS, as both of the corresponding hardware components incur energy overhead.

Components enabling fine-grained control over scaling also consume more energy. The operational voltage range of most MSP430-series MCUs, for example, is between 3.6 V and 1.8 V. In the absence of on-chip DVFS support, detecting and regulating even at every 100 mV drop in this range is impractical, as it would require a large number of accurate voltage detectors and narrow regulators.

Worse still, MSP430 clock registers can be configured to generate 4096 discrete clock frequencies in the range 1-16 MHz, each having its own specific operational voltage range. Since all such frequencies are not factory calibrated,

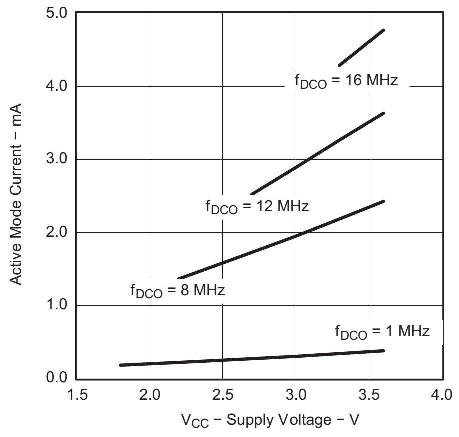


Figure 4: Voltage supply range and current consumption of factory calibrated frequencies.

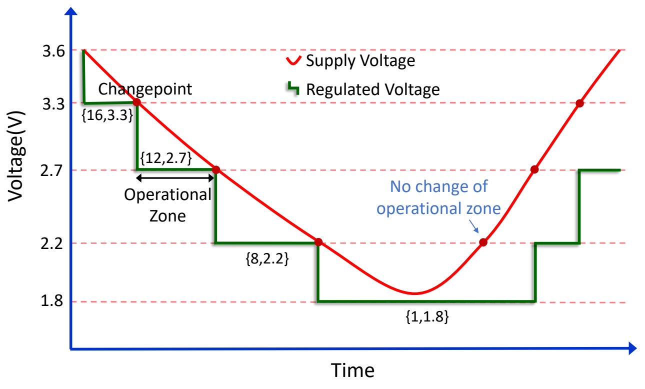


Figure 5: Working of D^{2}VFS. *At each changepoint, the D^{2}VFS driver determines whether or not place the MCU in a new operational zone* {*f* (*MHz*), *V*}.

one must empirically calibrate them for each MCU, which is a daunting task, and precisely derive their relationship with an MCU supply voltage to safely scale at run-time.

We therefore restrict frequency scaling in D^{2}VFS to the four factory calibrated frequencies, i.e., 1 MHz, 8 MHz, 12 MHz and 16 MHz. This still gives us enough leverage to scale the MCU clock speed for energy efficiency, while keeping the associated overhead low. For example, we demonstrate that four voltage changepoint detectors and regulation levels (at 3.6 V, 3.3 V, 2.7 V, and 2.2 V) suffice to effectively achieve DVFS on intermittently-computing devices.

The factory calibrated configurations for these frequencies are permanently burnt into the on-chip flash, thus, enabling direct reconfiguration of the MCU speed at run-time by merely writing these configurations into the clock registers. Their operational voltage range and power consumption are also well understood, as shown in Fig. 4, and illustrated in the data sheet [45]. This also spares the developers from tedious calibration efforts.

Operation. Fig. 5 explains the working of D^{2}VFS. Say the active epoch begins at a capacitor voltage of 3.6 V. The MCU is thus configured to run at 16 MHz and the supply voltage is regulated at 3.3 V, which is the minimum voltage required to operate with this frequency.

Changepoint	**Operational Zone** {f (MHz), V}
3.6	{16, 3.3}
3.3	{12, 2.7}
2.7	{8, 2.2}
2.2	{1, 1.8}

Table 1: D^{2}VFS changepoint decisions.

In the absence of energy harvested from the ambient, the capacitor voltage continuously drops until 3.3 V, i.e., the first changepoint, where an interrupt fires. The D^{2}VFS driver scans and compares the current state of voltage detectors with the previous one to determine if the current changepoint is reached due to dropping or rising capacitor voltage. In the former case, we place the MCU into a new operational zone by scaling down the MCU speed to 12 MHz and reconfiguring the voltage regulators to deliver 2.7 V to the MCU. A similar scale-down process applies at subsequent changepoints if they are triggered by a dropping voltage due to a capacitor discharge.

If a changepoint is reached from the opposite direction, namely, due to a capacitor charge, we hold back scaling up of the operational zone until the immediately higher changepoint is reached, as shown in Fig. 5. This effectively implements an hysteresis that prevents dangerous oscillations around a changepoint.

Suppose, for example, a changepoint is reached due to capacitor recharge, as in the case of the changepoint at 2.2 V on the right of Fig. 5. If the D^{2}VFS driver immediately places the MCU in the new operational zone, that is, it transitions from {1 MHz, 1.8 V} to {8 MHz, 2.2 V}), a sudden discharge of the capacitor is likely to happen due to an increase in power consumption. This may result in the same changepoint to be hit from the opposite direction, triggering a scale down of the MCU to the previous operational zone. In the worst case, this may result in oscillating behaviors.

This technique may result in the MCU operating in different operational zones for the same capacitor voltage range, depending on whether it is being entered due to voltage drop or voltage rise. As shown in Fig. 5, the MCU operates at {8 MHz, 2.2 V} when the capacitor voltage in the range 2.2 V-2.7 V in the case it is entered via the higher changepoint at 2.7 V, but it operates at {1 MHz, 1.8 V} in the opposite case. Tab. 1 summarizes the scaling decisions.

The D^{2}VFS driver is also in charge of *sequencing* of transitions across changepoints. When scaling up, the voltage regulator is reconfigured first to deliver a higher voltage before increasing the frequency. When scaling down, frequency is reduced first, then voltage is decreased.

D^{2}VFS derives its name—discrete dynamic voltage and frequency scaling—from its voltage regulation behavior that mimics a *discrete* step function, as shown in Fig. 5. A conventional frequency scaling approach may allow the voltage to traverse the entire supply range of a given frequency.

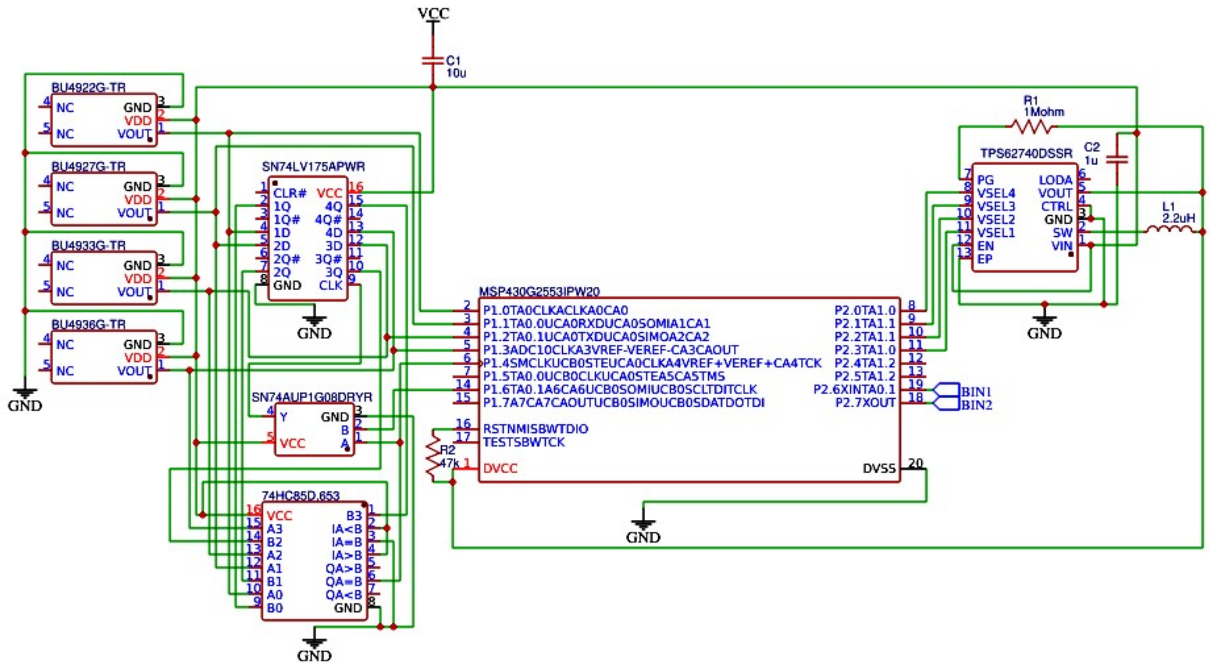


Figure 6: D^{2}VFS schematic implementation in EasyEDA.

3.2 Implementation

We design hardware support for D^{2}VFS in EasyEDA. Fig. 6 shows the schematics of our implementation. The capacitor voltage is intercepted by BU49xx-series voltage detectors [41]. Every detector generates a 1 if the capacitor voltage is above their detection level, 0 otherwise. The output from the detectors is used as an input to a trigger circuit, which fires an interrupt whenever it detects a change in the input, namely, a changepoint is reached.

The interrupt triggers the execution of D^{2}VFS driver, which scans the state of voltage detectors at GPIOs. This is needed to determine the direction the changepoint is reached and to take corresponding scaling decisions. We change the MCU operational zone by reconfiguring the clock registers and voltage regulators, which are available at GPIOs.

Implementation of D^{2}VFS is enabled by highly efficient voltage detection and regulation ICs. Below we provide important details regarding these amid justifying our choices.

Voltage detectors and regulators. We use BU49xx series ROHM semiconductor low-voltage standard CMOS voltage detectors [41]. These are known for high-accuracy ($\pm 1\%$) and ultra-low current consumption. Their detection range is 0.9-4.8 V at 0.1 V steps, and they can operate in the temperature range -40 °C to 125 °C. Tab. 2 shows the salient features of detector ICs we use in our schematic.

Detector	Detection Voltage			Circuit Current
	Min	Typ.	Max	μA
BU4922	2.19	2.2	2.22	0.26
BU4927	2.67	2.7	2.73	0.29
BU4933	3.27	3.3	3.33	0.34
BU4936	3.56	3.6	3.64	0.35

Table 2: Salient features of D^{2}VFS voltage detectors.

We use TI's TPS62740x step down converters for low power applications [47]. Their input voltage range is 2.2-5.5 V and they can regulate up to 16 output voltages between 1.8-3.3 V with a step size of 100 mV. If the input voltage falls below 2.2 V, the converter enters no ripple (or bypass) mode where it stops regulation and the output is directly connected to the input voltage. Because of that, the MCU obtains unregulated supply voltage for the range 2.2 to 1.8 V, i.e., the lowest operational zone of D^{2}VFS. The component offers 90% efficiency for up to 300 mA output current, which is way beyond the maximum current consumption of MSP430 MCUs, and draws 360 nA quiescent current.

Interrupts. D^{2}VFS needs to generate an interrupt whenever a changepoint is detected. The interrupt logic compares the previous state of detectors with the current one and fires an interrupt when they differ. We use TI's SN54LV175A Quadruple D-TYPE flipflops [36] to store the previous detector state and Nexperia's 74HC85 4-bit magnitude comparators [37]. Both satisfy our supply voltage and ultra-low power requirements. Additionally, we employ a SN74AUP1G08 2-input positive-AND gate [46], with MCU clock and comparator output at its input to clock flipflops. This is to ensure that we do not miss any change in detectors' state if they occur at the same time as the clock pulse.

Altogether, these components increase the energy consumption of the MCU between 0.56% to 11.5% depending on the capacitor size and operational zone, while offering substantial scaling benefits.

We thoroughly validate our schematic by generating all possible voltage inputs to measure the outputs of both voltage detectors and regulators as well as the power dissipation of the corresponding ICs. We use these measurements to emulate and benchmark the performance of D^2VFS.

4 Benchmark Evaluation

We evaluate the performance of D^2VFS using a combination of three benchmarks across two system supports. Our results indicate that using D^2VFS turns into:

- up to 9× increase in clock cycles per active epoch, increasing the amount of computation performed within a single capacitor charge;
- up to one-sixth smaller energy buffer to complete the same workload, cutting the time to reach the operating threshold voltage and enabling smaller device footprints;
- a two-fold improvement in the number of required checkpoints, as we allow the system to complete a larger portion of the workload before a checkpoint is required;
- up to 300% shorter completion times for a given workload, increasing system's responsiveness and despite the additional overhead of D^2VFS.

In the following, Sec. 4.1 describes the settings, whereas Sec. 4.2 to Sec. 4.4 discuss the results.

4.1 Settings

We describe the benchmark applications, the hardware and software we use for experiments, the metrics we compute, and the power profiles we test.

Benchmarks. We consider three benchmarks widely employed in intermittent computing [22, 40, 7, 48]: *i)* a Fast Fourier Transform (FFT) implementation, *ii)* RSA cryptography, and *iii)* Dijkstra spanning tree algorithm. FFT is representative of signal processing functionality in embedded sensing. RSA is an example of security support on modern embedded systems. Dijkstra's spanning tree algorithm is often found in embedded network stacks [21].

These benchmarks offer a variety of memory access patterns and processing load. The FFT implementation has moderate processing requirements; RSA demands great MCU resources; Dijkstra's algorithm only handles integer data and has the lightest processing demands. This diversity allows us to generalize our conclusions. The implementations are from public code repositories [34].

Systems and platforms. We measure D^2VFS performance using established system support for intermittent computing.

In Hibernus [7], an interrupt is fired whenever the capacitor voltage falls below a statically-defined threshold, which prompts the system to take a checkpoint exactly at that moment. Using MementOS [40], the code is instrumented by inserting checkpoint calls at determined locations, for example, after loops or function calls. Upon reaching any of these calls, a checkpoint takes place if the capacitor voltage is found below a threshold obtained using repeated emulation experiments and user-provided energy traces. Checkpoint operations in MementOS are more energy efficient, as it only copies the *allocated* regions of main memory on non-volatile storage, compared to Hibernus that dumps the entire main memory regardless of content.

We consider as baselines the execution of the aforementioned benchmarks along with these support systems on a *statically configured* MSP430G2553 running at 1 MHz, 8 MHz, 12 MHz, and 16 MHz [45]. To make our analysis of MementOS independent of specific energy traces, we manually sweep the possible parameter settings at steps of 0.2 V and use the best performing one [40].

We must note, however, that the choice of underlying system support for evaluating D^2VFS is not critical. We choose these systems because of their widespread use for benchmarking intermittently-computing systems [2, 12, 15, 31]. The ability of D^2VFS to place the MCU in the most efficient operational zone should, in principle, equally benefit any system support, whether based on checkpointing [12] or abstractions with transactional semantics [31, 15, 33].

Metrics. We compute four key metrics to study the various trade-offs involved:

- The *cycles per active epoch* is the number of clock cycles achieved in a single active epoch, whose duration is determined by power consumption, capacitor size, and shutdown voltage threshold. This figure represents a measure of the amount of computation enabled in an active epoch.
- The size of the *smallest energy buffer* is the minimum energy to complete a workload. If too small, a system may be unable to make progress and complete checkpoints, causing non-termination. Using capacitors, however, smaller sizes reach the operating voltage sooner and enable smaller device footprints.
- The *number of checkpoints* is the total number of checkpoints to complete a workload. The more are necessary, the more the system subtracts energy from useful computations. Increasing clock cycles in an active epoch allows the application to progress further on the same charge.
- The *completion time* is the time to complete a workload, excluding deployment-dependent recharge times. D^2VFS gradually de-accelerates the MCU and introduces a run-time overhead. On the other hand, due to availability of more compute cycles, a larger portion of workload can be completed within a single active epoch.

Power profile and measurements. We use a foundational power profile found in existing literature [22, 10, 48, 40] that provides fine-grained control over executions and facilitates interpreting results. The device boots with the full capacitor and computes until the capacitor is empty again. In the mean time, the environment provides no additional energy. Once the capacitor is empty, the environment provides new energy until the capacitor is full again and computation resumes.

This profile is also representative of a staple class of intermittently-powered applications, namely, those based on wireless energy transfer [11, 14]. With this technology, devices are quickly charged with a burst of wirelessly-transmitted energy until they boot. Next, the application runs until the capacitor is empty again. The device rests dormant until another burst of wireless energy comes in.

We trace the executions on a MSP430G2553 Launchpad interfaced with an FRAM chip for persistent storage. For this purpose, the Launchpad offers a range of hooks, en-

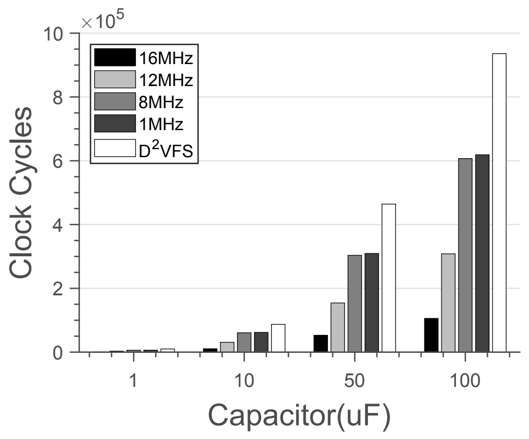


Figure 7: Cycles per active epoch. *D^2VFS retains its benefits over static frequency settings despite the implementation overhead due to additional hardware and the D^2VFS driver.*

abling fine-grained measurements. For example, it allows us to practically implement frequency scaling and ascertain the time taken and power consumed by every operation at different frequencies amid accounting for the D^{2}VFS overhead. We instrument our benchmarks to collect these measurements and simulate energy draw from the capacitor during execution. Due to their extensive usage in electronics, the accuracy of capacitor models for their charge-discharge behavior and voltage drop between the plates is well established and undisputed [19]. The results are obtained from 100 application iterations.

4.2 Computation per Active Epoch

We compare the number of clock cycles enabled by D^{2}VFS, which represents the amount of computation available in an active epoch, with the statically configured factory-calibrated frequency settings. We use variable capacitor sizes, thus different energy storage capacities, and record the clock cycles achieved in a single charge of the capacitor.

Fig. 7 shows the results. These are similar to the first order approximation shown in Sec. 2 to scrutinize D^{2}VFS. Nonetheless, we observe that the margin of improvement is only slightly reduced compared to Fig. 2, validating the energy efficiency of the concrete D^{2}VFS implementation. An appropriate selection of scaling granularity and our choice of hardware components ensures that the energy overhead of D^{2}VFS components is kept low, and that the advantages of dynamic voltage and frequency scaling are retained.

Worth noticing is also that the margin of improvement increases with the size of capacitor. This is due to constant power dissipation of D^{2}VFS components and static number of changepoints in an active epoch irrespective of capacitor sizes. Thus, the percentage energy overhead of D^{2}VFS decreases when the capacitor size increases.

4.3 Checkpoints and Energy Buffers

The results in the number of checkpoints and in the size of the smallest energy buffer are intimately intertwined.

Fig. 8 depicts the reduction in the number of checkpoints against variable capacitor sizes. A portion of these charts only shows the performance of the D^{2}VFS-based execution, as the ones with static frequency settings are unable to complete the workload with too small capacitors. When a comparison is possible, the improvements for D^{2}VFS are substantial and apply across benchmarks and capacitor sizes.

The number of checkpoints is maximum at 16 MHz. There, the MCU only operates in a narrow interval of supply voltage, resulting in fewer clock cycles in an active epoch compared to other static frequency settings. Therefore, the execution occurs in small bursts separated by checkpoints.

Fig. 9 reports the minimum size of the capacitor required to complete the given workloads. A D^{2}VFS-equipped system constantly succeeds with smaller capacitors. With Hibernus, D^{2}VFS allows one to use a capacitor up to one-sixth of the one required with a static frequency setting. Since checkpoints in MementOS are more energy efficient, it generally achieves smaller capacitor sizes compared to Hibernus. However, the heuristics employed by MementOS[2] for inserting checkpoint calls are not always productive and may lead to an increase in peak energy demand despite energy efficient checkpoint operations. For example, in one particular instance of RSA, the calls to checkpoint fell so far apart that the energy needed to guarantee successful execution exceeds that of Hibernus, requiring an even bigger capacitor size, as shown in Fig. 9b.

These results are directly enabled by the ability of D^{2}VFS to increase the number of clock cycles, thus allowing systems to perform more computation in a single active epoch. Applications thus progress farther on a single charge, while reducing the number of checkpoints en route to completion. Similarly, the smallest amount of energy the system needs to have available at once to move from one checkpoint to the next without any power failure in between, reduces as well. Smaller capacitors mean reaching operating voltage faster and smaller device footprints.

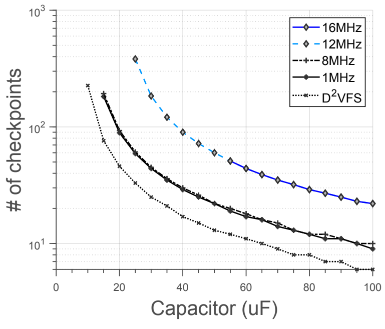
These results demonstrate that D^{2}VFS allows intermittently-computing systems to reduce the total *time* invested in checkpoint operations because of a reduction in their number. This leads to shorter completion times for given workloads, as we investigate next.

4.4 Completion Time

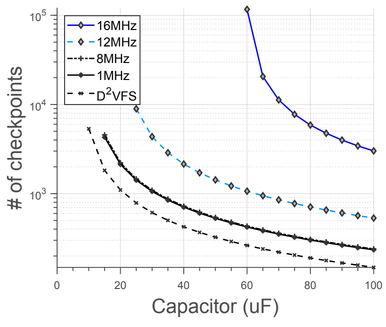
D^{2}VFS gradually de-accelerates the processor to exploit the maximum range of the MCU supply voltage. On the one hand, this increases the instruction execution time compared to higher frequencies (e.g., 16 MHz). On the other hand, checkpointing operations are delayed and more useful progress is made before a checkpoint is eventually required. We explore the relative contribution of either aspect to the total completion time for a given workload.

Fig. 10 reports the results. We execute these experiments using the smallest common capacitor needed to guarantee completion with a given static frequency setting and D^{2}VFS, as these are the preferred settings for an intermittently-computing device. We can see that the reduction in speed is not only compensated, but actually overturned by the ability of D^{2}VFS to complete larger portions of the workload

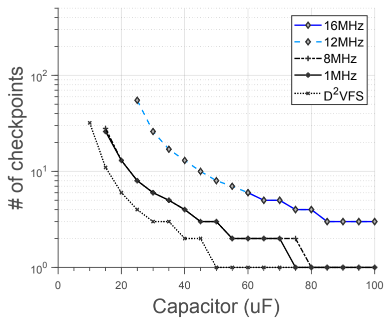
[2]MementOS suggests two strategies for inserting checkpoint calls: (1) after every loop, or (2) after return from every function call [40].



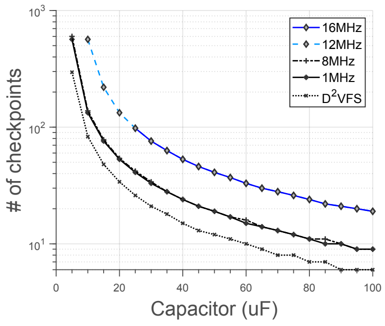
(a) Hibernus (FFT)



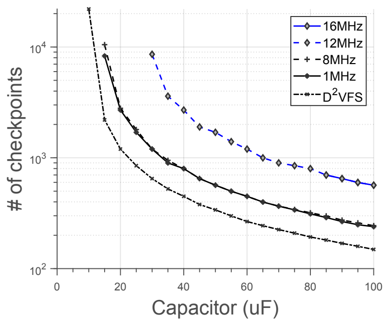
(b) Hibernus (RSA)



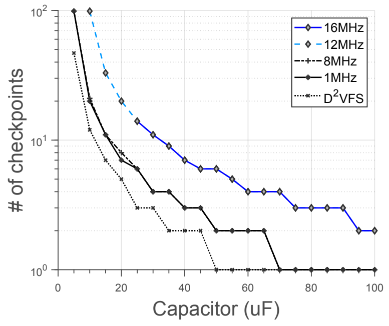
(c) Hibernus (Dijkstra)



(d) MementOS (FFT)

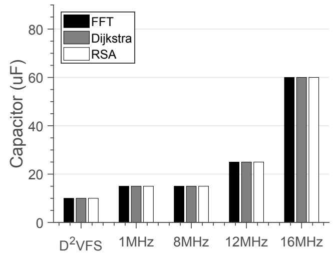


(e) MementOS (RSA)

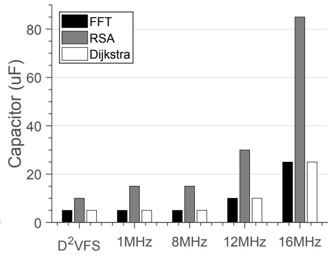


(f) MementOS (Dijkstra)

Figure 8: Number of checkpoints necessary against varying capacitor sizes. *D^{2}VFS allows the system to progress farther on a single charge, reducing the number of checkpoints needed to complete a workload.*



(a) Hibernus



(b) MementOS

Figure 9: Smallest capacitor. *A D^{2}VFS-equipped system completes the workload with smaller capacitors. This is due an increase in available clock cycles in a single active epoch.*

in each setting before initiating checkpoints. This allows D^{2}VFS to complete the workload much earlier, increasing the system's responsiveness.

Comparing across static frequency settings is not possible in these experiments because the respective smallest capacitor sizes ensuring completion of their workload are different, as shown in Fig. 9. With the change in capacitor size, the factors that impact completion time such as the amount of energy storage, the number of available clock cycles in a single active epoch and, more importantly, the location and the number of checkpoints change drastically. Similarly, the reduced energy overhead in MementOS due to smaller size of checkpoints causes it to produce a different outcome using static frequency settings compared to Hibernus.

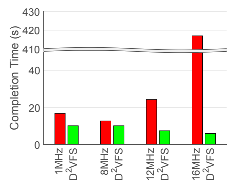
Regardless, D^{2}VFS consistently and substantially outperforms all static frequency settings across all benchmarks, and its benefits are not affected by these factors.

5 Real World Evaluation

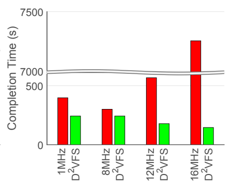
The benchmark evaluation of D^{2}VFS uses a synthetic power profile that does not take into account the recharge times of the capacitor. In real world settings, the recharge times may differ depending upon the frequency settings. A capacitor retains large amounts of residual energy at higher frequencies, since these only operate in a narrow interval of supply voltage, but drains off considerably at lower frequencies. Conversely, the charging rate of capacitor is faster at the start but then tapers off as the capacitor acquires additional charge at a slower rate.

We thus investigate the impact of these conflicting factors using real-world power traces and confirm whether the results of Sec. 4 carry over to real world settings. We build the same activity recognition (AR) application often seen in the literature [13, 40, 31, 15, 33], using the same source code [17]. The rest of the setup is as for Hibernus in Sec. 4.

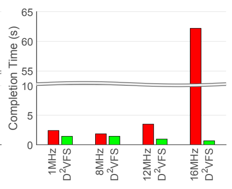
We focus on completion time using the smallest capacitor that ensures completion of the AR application across D^{2}VFS and the baselines we consider. Based on the results of Sec. 4, completion time is indeed the one metric where all the involved trade-offs manifest.



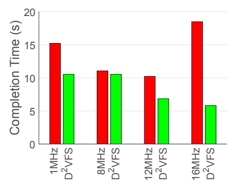
(a) Hibernus(FFT)



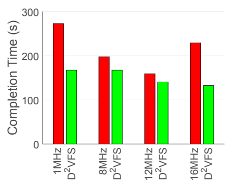
(b) Hibernus (RSA)



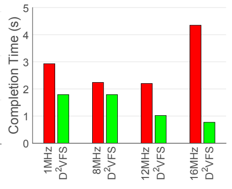
(c) Hibernus (DIJ)



(d) MementOS (FFT)



(e) MementOS (RSA)



(f) MementOS (DIJ)

Figure 10: Completion time. *The run-time overhead due to D^{2}VFS is overturn by increasing the number of clock cycles available, and therefore the amount of computation possible in a single active epoch, and thus reducing the number of checkpoints. The combined effect shortens completion times.*

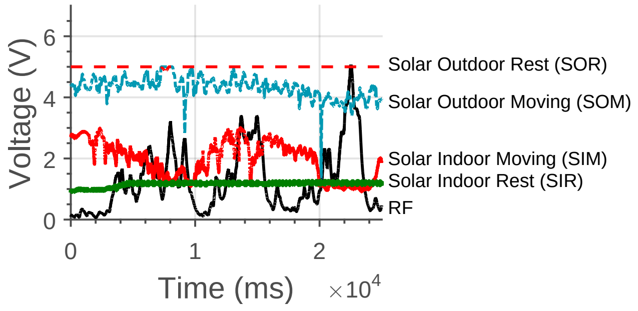


Figure 11: Voltage traces used for evaluation.

Power traces. We consider five power traces, obtained from diverse energy sources and in different settings. One of the traces is the RF trace from MementOS [40, 26]. The black curve in Fig. 11 shows an excerpt, plotting the instantaneous voltage reading at the energy harvester over time.

We collect four additional traces using a mono-crystalline solar panel [44] and an Arduino Nano [4] to measure the voltage output across a 30 kΩ load, roughly equivalent to the resistance of the MSP430G2553 in active mode [45]. Using this setup, we experiment with different scenarios. We attach the device to the wrist of a student to simulate a fitness tracker. The student roams in the university campus for outdoor measurements (SOM), and in research lab for indoor measurements (SIM). Alternatively, we keep the device on the ground right outside the lab for outdoor measurements (SOR), and at desk level in our research lab for the indoor measurements (SIR).

Fig. 11 demonstrates the extreme variability and considerable differences among the power traces we consider.

Results. Fig. 12 shows the completion times across all traces. D^{2}VFS consistently performs better, regardless of the power trace and static frequency setting it is compared with. Another notable observation is the failure of MCU to complete the workload when running at 1 MHz with the RF trace and at 16 MHz with both the RF and SIR traces. Both the *lowest* and *highest* frequencies get penalized. The former happens because of the highest energy per clock cycle ratio as well as by a considerably drained off capacitor at the end of each active epoch; whereas the latter is to endure the largest number of costly checkpoints.

The intermediate frequencies, that are, 8 MHz and 12 MHz, perform better comparatively, but D^{2}VFS reaps the highest benefit from all possible frequency settings; it makes the system operate in the most efficient configuration given a certain capacitor voltage, while also reducing the checkpointing overhead. Based on these results, the performance and trade-offs we discuss in Sec. 4 are thus confirmed with a concrete application and diverse power traces.

6 Related Work

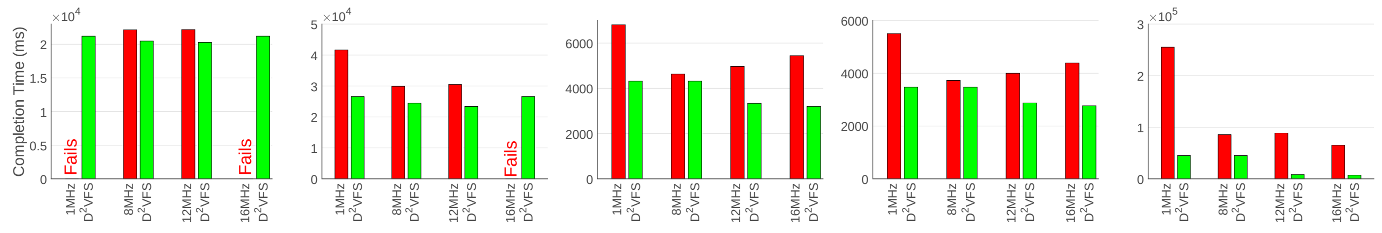
A large body of work investigates DVFS in mainstream computing. However, to scale down our discussion on related works, here we focus on DVFS in embedded systems. We broadly divide related literature in three categories: general purpose embedded systems, wireless sensor networks, and intermittently-computing devices.

Embedded systems. Salehi et al. [42] present an adaptive voltage and frequency scaling technique that rapidly tracks the workload changes to meet soft real-time deadlines. Their work demonstrates considerable power savings and fewer frequency updates compared to DVFS systems based on fixed update intervals. HyPowMan [9] considers the problem of power consumption minimization for periodic real-time tasks that are scheduled over multiprocessor platforms with dynamic power management (DPM) and DVFS capabilities. This technique takes a set of well-known existing DPM and DVFS policies, each performing well for a given set of conditions, and adapts at runtime to the best-performing policy for any given workload.

Huang et al. [20] apply DVFS to mixed-criticality systems, and show that DVFS can be used to help critical tasks meet deadlines by speeding up the processor when they are bound to miss the deadline. Liu et al. [30] employ DVFS to optimize system thermal profile, to prevent run-time thermal emergencies, and to minimize cooling costs. They present a framework for system designers to determine a proper thermal solution and provide a lower bound on the minimum temperature achievable by their DVFS technique. RT-DVFS [38] targets embedded operating systems, such as in mobile phones and camcorders. It modifies the OS's real-time scheduler and task management service to provide significant energy savings while maintaining real-time deadline guarantees. Generalized Shared Recovery (GSHR) [49] efficiently uses DVFS techniques to achieve a given reliability goal for real-time embedded applications.

These works provide foundational knowledge on applying DVFS in embedded systems, yet their design goals are very different, and their techniques are not directly applicable to intermittently-computing devices.

Wireless sensor networks. Kulau et al. [23, 24, 25] thoroughly analyze the effects of undervolting for a typical wireless sensor node both in theory and practice. They show



(a) RF (b) SIR (c) SOM (d) SOR (e) SIM

Figure 12: Performance of the AR app running on Hibernus. *Performance gains with D^2VFS are observed across diverse power traces obtained from different energy sources. The average improvement compared to static frequency settings ranges from 30% to 300% across all traces.*

that a wireless sensor network can still work reliably, even if the voltage recommendations are violated, because there is a correlation between temperature and error-proneness at the same voltage level, and that ideal voltage levels depend on environmental conditions. Powell et al. [39] design DVFS hardware to meet battery life and form factor expectations of body area sensor networks. Similar to this are the works on developing DVFS techniques in distributed microsensor networks [35] and in sensing applications with deadlines [3].

Many of these works are similar to ours in spirit, as they all aim to conserve energy, yet these approaches consider battery-powered devices with *finite* energy supplies, and tend to accept performance penalties to increase life time. On the contrary, we deal with intermittent but unbounded energy supplies where the goal is to increase the amount of work done in an active epoch that, in turn, improves a number of other key performance metrics. The techniques we use are rather aggressive compared to the ones employed in wireless sensor networks in scaling both voltage and frequency.

Intermittently-computing devices. EA-DVFS [29] is a high-level simulation-based study that highlights the benefits of DVFS in achieving real-time operation on battery-less devices. As the corresponding hardware architecture and implementation is not available, this approach cannot be used as baseline in our work. Noise-aware DVFS sequence optimization techniques are proposed to reduce noise, i.e., extra current that accompanies the clock speed transition, in energy-harvesting devices [32]. This work is complementary to our efforts and, if integrated with them, we expect a further reduction in the energy overhead.

Lin et al. [28] model a framework for concurrent task scheduling and dynamic voltage and frequency scaling in real-time embedded systems with energy harvesting. They develop a global controller that performs optimal operating point tracking for the PV panel, state-of-charge management for the supercapacitor, and energy-harvesting aware real-time task scheduling with DVFS. Li et al. [27] also provide early insights into the benefits of jointly scaling workload, voltage, and frequency in multi-core sensor networks powered by energy harvesting.

These works provide useful early-stage insights on employing DVFS in energy harvesting devices. However, D^2VFS is the first concrete implementation of any such technique, along with a detailed evaluation that precisely highlights the benefits of employing DVFS in intermittently-computing devices.

7 Conclusion

The peculiar provisioning patterns of ambient energy harvesting, together with the features of modern low-power MCUs, create an opportunity to improve the energy efficiency of intermittently-computing devices by dynamically adjusting supply voltage and frequency settings.

We seized to the opportunity and presented D^2VFS, a hardware and software co-design that seeks to reap maximum benefits from these patterns and features by reducing the overhead imposed by additional hardware components and software drivers. To make up for the lack of dedicated hardware on low-power MCUs, we employ an external low-power DVFS engine to identify voltage changepoints and trigger the execution of a software driver that scales both supply voltage and frequency settings. The performance improvements of D^2VFS are notable. For example, the number of available clock cycles in a single active epoch is increased by 40-900% compared with static frequency settings. The ultimate impact is on workload completion times, which are reduced by up to 300%, as shown in real world settings with diverse power traces.

8 References

[1] S. Ahmed, A. Bakar, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola. The betrayal of constant power x time: Finding the missing joules of transiently-powered computers. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES, 2019.

[2] S. Ahmed, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola. Efficient intermittent computing with differential checkpointing. In *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, LCTES, 2019.

[3] R. Antonio, R. Costa, A. Ison, W. Lim, R. Pajado, D. Roque, R. Yutuc, C. Densing, M. T. de Leon, M. Rosales, and L. Alarcon. Implementation of dynamic voltage frequency scaling on a processor for wireless sensing applications. In *TENCON*, pages 2955–2960, 11 2017.

[4] ARDUINO. *NANO*, 2018. `https://store.arduino.cc/usa/arduino-nano` (accessed 2019-09-25).

[5] S. Arra, J. Leskinen, J. Heikkila, and J. Vanhala. Ultrasonic power and data link for wireless implantable applications. In *2nd International Symposium on Wireless Pervasive Computing*, 2007.

[6] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016.

[7] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters*, 2015.

[8] M.-M. U. R. battery-free device. *Farsens*. `http://www.farsens.com/en/products/medusa-m2233/` ((accessed 2019-09-25)).

[9] M. K. Bhatti, C. Belleudy, and M. Auguin. Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies. *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, pages 184–191, 2010.

[10] N. Bhatti and L. Mottola. Efficient state retention for transiently-powered embedded sensing. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks*, 2016.

[11] N. A. Bhatti, M. H. Alizai, A. A. Syed, and L. Mottola. Energy harvesting and wireless transfer in sensor network applications: Concepts and experiences. *ACM Transactions on Sensor Networks*, 2016.

[12] N. A. Bhatti and L. Mottola. HarvOS: Efficient code instrumentation for transiently-powered embedded sensing. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*, 2017.

[13] A. Branco, L. Mottola, M. H. Alizai, and J. H. Siddiqui. Intermittent asynchronous peripheral operations. 2019.

[14] M. Buettner, B. Greenstein, and D. Wetherall. Dewdrop: An energy-aware runtime for computational RFID. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation*, 2011.

[15] A. Colin and B. Lucia. Chain: tasks and channels for reliable intermittent programs. In *Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2016.

[16] C. Dagdeviren, P. Joe, O. L. Tuzman, K.-I. Park, K. J. Lee, Y. Shi, Y. Huang, and J. A. Rogers. Recent progress in flexible and stretchable piezoelectric devices for mechanical energy harvesting, sensing and actuation. *Extreme Mechanics Letters*, 2016.

[17] A. R. Group. *Benchmark Applications*, 2018. `www.github.com/CMUAbstract/releases#benchmark-applications` (accessed 2019-09-25).

[18] J. Hester and J. Sorber. Flicker: Rapid prototyping for the batteryless internet-of-things. In *Proceedings of the 15th ACM Conference on Embedded Networked Sensor Systems*, 2017.

[19] P. Horowitz and W. Hill. *The art of electronics*. Cambridge Univ. Press, 1989.

[20] P. Huang, P. Kumar, G. Giannopoulou, and L. Thiele. Energy efficient dvfs scheduling for mixed-criticality systems. In *Proceedings of the 14th International Conference on Embedded Software*, EMSOFT, pages 11:1–11:10, New York, NY, USA, 2014. ACM.

[21] O. Iova, P. Picco, T. Istomin, and C. Kiraly. RPL: The routing standard for the Internet of Things... Or Is It? *IEEE Communications Magazine*, 2016.

[22] H. Jayakumar, A. Raha, W. S. Lee, and V. Raghunathan. Quick recall: A HW/SW Approach for Computing across Power Cycles in Transiently Powered Computers. *ACM Journal on Emerging Technologies in Computing Systems*, 2015.

[23] U. Kulau, F. Büsching, and L. Wolf. Undervolting in wsns: Theory and practice. *Internet of Things Journal, IEEE*, 2:190–198, 06 2015.

[24] U. Kulau, F. Büsching, and L. Wolf. Idealvolting: Reliable undervolting on wireless sensor nodes. *ACM Trans. Sen. Netw.*, 12(2):11:1–11:38, Apr. 2016.

[25] U. Kulau, S. Rottmann, S. Schildt, J. Balen, and L. Wolf. Undervolting in real world wsn applications: A long-term study. In *DCOSS*, pages 9–16, 05 2016.

[26] P. Lab. *RF Trace*, 2018. `https://github.com/PERSISTLab/BatterylessSim/tree/master/traces` (accessed 2019-09-25).

[27] X. Li. Dynamic voltage-frequency and workload joint scaling power management for energy harvesting multi-core wsn node soc. *Sensors*, 17:310, 02 2017.

[28] X. Lin, Y. Wang, S. Yue, N. Chang, and M. Pedram. A framework of concurrent task scheduling and dynamic voltage and frequency scaling in real-time embedded systems with energy harvesting. *Proceedings of the International Symposium on Low Power Electronics and Design*, 35:70–75, 09 2013.

[29] S. Liu, Q. Qiu, and Q. Wu. Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting. In *Proceedings of the conference on Design, automation and test in Europe*, pages 236–241. ACM, 2008.

[30] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang. Thermal vs energy optimization for dvfs-enabled processors in embedded systems. In *Proceedings of the 8th International Symposium on Quality Electronic Design*, ISQED, pages 204–209, Washington, DC, USA, 2007. IEEE Computer Society.

[31] B. Lucia and B. Ransford. A simpler, safer programming and execution model for intermittent systems. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2015.

[32] S. Luo, C. Zhuo, and H. Gan. Noise-aware dvfs transition sequence optimization for battery-powered iot devices. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, pages 27:1–27:6, New York, NY, USA, 2018. ACM.

[33] K. Maeng, A. Colin, and B. Lucia. Alpaca: Intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages*, 2017.

[34] mbed. *IoT OS*, 2017. `goo.gl/u918jX`.

[35] R. Min, T. Furrer, and A. Chandrakasan. Dynamic voltage scaling techniques for distributed microsensor networks. In *Proceedings of the IEEE Computer Society Annual Workshop on VLSI (WVLSI'00)*, WVLSI '00, pages 43–, Washington, DC, USA, 2000. IEEE Computer Society.

[36] Mouser. *Data Sheet*, 2018. `http://www.ti.com/lit/ds/symlink/sn74lv175a.pdf` (accessed 2019-09-25).

[37] Mouser. *Data Sheet*, 2018. `https://eu.mouser.com/datasheet/2/916/74HC_HCT85-1597760.pdf` (accessed 2019-09-25).

[38] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP, pages 89–102, New York, NY, USA, 2001. ACM.

[39] H. C. Powell, A. T. Barth, and J. Lach. Dynamic voltage-frequency scaling in body area sensor networks using cots components. In *Proceedings of the Fourth International Conference on Body Area Networks*, BodyNets '09, pages 15:1–15:8, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[40] B. Ransford. 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*, 2011.

[41] ROHM. *Data Sheet*, 2015. `https://www.rohm.com/datasheet/BU4910F/bu48xxg-e` (accessed 2019-09-25).

[42] M. E. Salehi, M. Samadi, M. Najibi, A. Afzali-Kusha, M. Pedram, and S. M. Fakhraie. Dynamic voltage and frequency scheduling for embedded processors considering power/performance tradeoffs. *IEEE Trans. Very Large Scale Integr. Syst.*, 19(10):1931–1935, Oct. 2011.

[43] J. R. Smith, A. P. Sample, P. S. Powledge, S. Roy, and A. Mamishev. A wirelessly-powered platform for sensing and computation. In *Proceedings of the 8th International Conference on Ubiquitous Computing*, 2006.

[44] I. SolarMD. *SLMD481H08L*, 2018. `http://ixapps.ixys.com/` (accessed 2019-09-25).

[45] TI. *Data Sheet*, 2013. `http://www.ti.com/lit/ds/symlink/msp430g2553.pdf` (accessed 2019-09-25).

[46] TI. *Data Sheet*, 2016. `http://www.ti.com/lit/ds/symlink/sn74aup1g08.pdf` (accessed 2019-09-25).

[47] TI. *Voltage Regulator*, 2016. `http://www.ti.com/lit/ds/slvsb02b/slvsb02b.pdf` (accessed 2019-09-25).

[48] J. Van Der Woude and M. Hicks. Intermittent computation without hardware support or programmer intervention. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, 2016.

[49] B. Zhao, H. Aydin, and D. Zhu. Generalized reliability-oriented energy management for real-time embedded applications. In *Proceedings of the 48th Design Automation Conference*, DAC, pages 381–386, New York, NY, USA, 2011. ACM.