Silent Stores in the Battery-less Internet of Things: A Good Idea?

Weining Song*, Stefanos Kaxiras*, Luca Mottola*†, Thiemo Voigt*, Yuan Yao*
*Uppsala University; †Politecnico di Milano
weining.song@angstrom.uu.se; stefanos.kaxiras@it.uu.se; luca.mottola@angstrom.uu.se;
thiemo.voigt@angstrom.uu.se; yuan.yao@it.uu.se

Abstract

We present experimental results investigating the use of silent stores in the battery-less Internet of Things (IoT). Silent stores occur in a program when the value being written onto memory exactly matches the memory content; general-purpose computing systems exploit silent stores to improve memory throughput. Battery-less IoT devices, on the other hand, rely on ambient energy harvesting as the only power source. Erratic energy patterns, however, cause frequent power failures, rendering executions *intermittent* and thereby requiring the use of energy-hungry non-volatile memory (NVM) to persist program states. The question we seek to answer is whether intermittently-computing IoT devices may reap any benefit from silent stores – or from a related variation called temporary silent store – as a way to save energy by sparing NVM operations. Our results point to a negative answer. Albeit *in principle* we observe copious (temporary) silent stores in staple battery-less IoT benchmarks, resource limitations of IoT devices and the features of modern NVM technology, such as FRAM, largely neutralize their impact on the energy figures *in practice*. In actual executions, for example, we measure a mere 2.2% energy consumption improvement, on average. The (negative) results we present here, obtained based on common IoT architectures such as ARM Cortex M* and MSP430 microcontrollers, raise awareness on the features of battery-less IoT devices and inform future research efforts.

Categories and Subject Descriptors

Embedded and cyber-physical systems [**Embedded systems**]: Embedded hardware

General Terms

Computer architecture, embedded systems

Keywords

Energy harvesting, non-volatile memory

1 Introduction

Limitations in a device physical footprint and maintenance concerns motivate replacing traditional batteries with ambient energy harvesting [5]. However, ambient energy is generally erratic, causing frequent and unanticipated power failures. For example, harvesting energy from RF transmissions to compute a simple cyclic redundancy check (CRC) may lead to 16 power failures over a six second period [5]. Executions thus become *intermittent*, as they consist of intervals of active computation interleaved by periods of recharging energy buffers [12].

Question. Due to resource constraints of IoT devices, energy failures normally cause a device to lose the program state. To ensure forward progress across energy failures, a variety of techniques exists that make use of *persistent state* stored on non-volatile memory (NVM) [12]. Persistent state is retrieved from NVM when energy is back, so devices resume close to the point of energy failure rather than performing a complete reboot. However, NVM causes significant energy overhead.

Reducing memory writes on NVM is potentially one way to abate this overhead. To do so, we investigate whether *silent* and *temporary silent stores* may bring any benefit in intermittent executions. Silent stores occur when the value being written matches the exact value already stored at the same memory location [15]. Temporary silent stores only change the value temporarily, to subsequently return to the previous value at the same memory location [16]. In general-purpose computing, works exist that eliminate (temporary) silent stores to improve memory throughput [13–16, 24].

With energy as the key performance metric and with different staple benchmarks, the question stands as to whether (temporary) silent stores are beneficial also for battery-less IoT devices that compute intermittently. Eliminating (temporary) silent stores, which avoid unnecessary write operations on NVM, is orthogonal to many existing intermittent computing solutions and may potentially boost their performance.

Our answer. The key contribution of this paper is to provide experimental evidence that, due to a combination of factors, the abundance of silent stores in staple benchmarks of battery-less IoT devices *does not* translate into matching energy savings. We argue that quantifying this performance is essential to make any solid claim on the potential applicability of silent stores and combat a case of hindsight bias [27].

We consider common IoT architectures such as ARM Cortex M* and MSP430 microcontrollers (MCUs), using a configuration where program variables are mapped to NVM-based main memory at compile time and only the register file must be saved before an energy failure to ensure forward progress.

Figure 1. Road-map through the paper.

Note that this configuration should, in principle, amplify the advantages due to sparing NVM operations.

Fig. 1 qualitatively depicts the conceptual and technical process we go through to investigate the use of silent stores in battery-less IoT devices. After summarizing background information and related work in Sec. 2, Sec. 3 illustrates the experimental setup, benchmarks, and baselines. We observe that store instructions abound in the benchmarks we consider. Of the entire number of store instructions in a benchmark, make that a 100% figure, we cannot know beforehand, however, how many of those are (temporary) silent.

Using both static code analysis and emulation experiments, in Sec. 4 we analyze store instructions in the benchmarks we consider and find that, *in principle*, silent stores account for nearly 45% of all store instructions. These figures, however, do not consider that the ability to capture (temporary) silent stores at run-time is limited by resource constraints germane to battery-less IoT devices. For example, when using a limited-size store buffer, only 23% of the total number of store instructions can be recognized as (temporary) silent. Hence, more than half of the potential benefit is already shaved off.

Next, in Sec. 5 we compute the energy saving that eliminating (temporary) silent stores may produce. This is the net performance gain we are ultimately interested in. The features of modern NVM technology, however, lead us to conclude that, *in practice*, we may only obtain a mere 2.2% energy improvement. This is specifically a result of the word-level addressing that NVM technology such as FRAM offers, combined with extremely low read/write energy costs and the symmetry in energy consumption between read and write operations.

We further discuss the key results of this paper in Sec. 6 and finally conclude the paper in Sec. 7.

2 Background and Related Work

In this section, we provide the necessary conceptual background and briefly survey related works.

Intermittent computing. The erratic energy patterns of ambient energy harvesting motivate efforts to address the resulting unpredictable power failures. Most existing works [12] focus on how to make efficient use of NVM technology to persist intermediate program states, thus ensuring forward progress. Some solutions employ a form of checkpointing [2–4, 7, 20, 25, 30]. This consists in replicating the application state on NVM at specific points in the code, where it is retrieved once the system resumes with sufficient energy. Other approaches offer abstractions that programmers use to define and manage persistent state [9, 17, 19, 23, 31], targeting platforms where a slice of the main memory is already mapped to NVM [21, 29].

Regardless of the specific approach, modern NVM technologies, such as FRAM, tend to replace CMOS-based memory, such as Flash, due to the higher access speed and energy efficiency [6, 13, 29]. The limited size of the FRAM chips is usually not a problem in battery-less IoT devices, as the codebases are limited and data is rarely retained locally but most often eventually transmitted to a central collection point.

Num.	Access Type	Addr.	Value	Silent?
1	Store	0x42	0x00	N/A
2	Store	0xF1	0x94	N/A
3	Load	0x42	0x00	N/A
...	...	...	...	...
9	Store	0xF1	0x94	Silent
10	Store	0x42	0x16	N/A
11	Store	0x42	0x00	Temp. Silent

Figure 2. Example of silent and temporary silent stores.

Silent stores. Lepak and Lipasti first present the concept of silent store [15]: a store instruction that writes the exact value already at that memory location. Later, they extend the concept and define temporary silent stores, where a store instruction temporarily changes the value, yet a further store instruction restores the previous value at the same memory location [16].

Consider the example in Fig. 2. Provided no other store instruction exists between instruction 3 and 9 that targets address `0xF1` or `0x42`, a silent store occurs at instruction 9 because the value already at `0xF1` is overwritten with the same value. A temporary silent store occurs at instruction 11, as the original value `0x00` is written again in place of `0x16` that was temporarily stored at instruction 10.

More generally, we state that a silent store S occurs at the $i+j$-th instruction in the program when

$$f(x_n)_i = f(x_{n+1})_{i+j} \tag{1}$$

with $f(x_n)_i$ being the value that the n-th write operation that appears in the program as the i-th instruction stores at memory address x. This value equals the $n+1$-th write operation at the same memory address x that appears as the $i+j$-th instruction in the program. Note this also entails that no other store instruction exists that operates on memory address x between instruction $i+1$ and $i+j-1$ in the program.

Similarly, a temporary silent store S_t occurs at the $i+j+z$-th instruction in the program when

$$f(x_n)_i = f(x_{n+2})_{i+j+z} \neq f(x_{n+1})_{i+j} \tag{2}$$

that entails the $i+j+z$ instruction in the program is the $n+2$-th write operation to memory address x and rewrites the same value that the n-th write operation stored, whereas another (single) write operation exists in between these two that (temporarily) writes a different value, for some j and z.

Detecting silent stores. Existing research on silent stores focuses on general-purpose systems to improve memory throughput. Several approaches exist.

Many processors use a write-back cache to achieve higher execution performance. Silent stores may occur when a cache line writes back to main memory. Krishnamurthy et al. [14] employ an additional cache besides the original one to detect (temporary) silent stores. The additional cache acts as a mirror of main memory, allowing them to save up to 50% memory writes. When a cache eviction occurs, the evicted cache line is compared to the additional cache first. If they match, a silent store occurred and nothing is written to main memory. An additional cache, however, may cause an energy overhead that a battery-less IoT device likely cannot afford.

Similar considerations apply to approaches that inspect the value of the memory before issuing the actual write operation [15]. Before writing a value to a memory address, the

processor reads the value stored in memory and compares it to the value about to be written. If they are the same, a silent store occurred and nothing is written to main memory. As a result, every store instruction converts to a read, compare, and possible-write operations. This yields an 11% execution speedup in mainstream platforms [15], but likely causes an energy overhead that is not practical for battery-less IoT devices.

Silent stores may also be detected by analyzing the code at compile time. Store instructions that may be definitely identified as silent may be removed altogether from the code, sparing the cost of fetch, execute, and write operations. This holds the potential of saving energy rather than causing an overhead like the approaches above; however, the lack of run-time information may drastically limit the ability of any static analysis to identify the full set of (temporary) silent stores.

Finally, store buffers may be employed to detect (temporary) silent stores. Store buffers are broadly used in modern processors to improve execution performance [11, 26]. Checking the addresses and values of store instructions in the store buffer is efficient, also in terms of energy consumption [15]. The limiting factor is the size of the store buffer, which is comparatively limited compared to both the distance between the store instructions possibly involved in a (temporary) silent store, and w.r.t. the number of different addresses that store instructions in a program may target.

The performance gains unlocked by silent stores in general-purpose computing prompt us to investigate their applicability in the battery-less IoT, despite the change in technology and performance metrics, filling a gap in the existing literature.

3 Setting

We describe the experimental setup we use to obtain the quantitative results we discuss in the rest of the paper.

Processor architectures and tools. We consider the two most common processor architectures for battery-less IoT devices: TI's MSP430 and ARM Cortex M*. Both are widely used in research prototypes [12] and real deployments [1].

We consider a 64KB main memory space in ARM Cortex M* architecture and a 16KB main memory space in the MSP430 architecture. We implement main memory with FRAM technology, which provides state-of-the-art performance in energy consumption and is vastly employed to support intermittent executions [2, 12, 20, 22, 30]. These architectures act as baselines and originally lack the ability to eliminate silent stores. When using store buffers, we consider the latter implemented with SRAM technology.

Note that for the purpose of this study, we extend both architectures with features that they currently do not have; specifically, we extend the MSP430 architecture with a store buffer and attach a FRAM-backed main memory to the Cortex M* architecture. In modern embedded processors, store buffers typically accommodate from 1 to 4 entries. We extend the store buffer size up to 16 entries in our experiments to investigate the trends in the number of silent stores it can capture.

We run representative benchmarks, described next, using the Gem5 [8] computer architecture simulator for ARM Cortex M* processors and the MSPSim [10] instruction-level emulator for MSP430 processors. To study the energy consumption of FRAM, we take energy figures from the MSP430FR5969 datasheet [29]. We build an energy model to quantify the energy consumption of each individual memory access on FRAM. By integrating this model into Gem5 and MSPsim, we accurately calculate the total energy consumption associated with accessing FRAM during program execution. Note how built-in FRAM in the latter shows the same energy cost for read and write operations [29], unlike the systems considered in previous literature [28]. We anticipate that the symmetry in energy cost between read and write operations is one of the causes for the results we report in Sec. 5.

Table 1. Memory operations in the benchmarks we consider depending on processor architecture.

Benchmarks	MSP430		ARM	
	Loads	Stores	Loads	Stores
rsa	27651	3088	46351	5514
crc	22227	805	36912	2232
aes	551023	52936	247017	29970
basicmath	2454184052	300706881	95168101	10786063
fft	62713698	7781205	4476449	502436
patricia	N/A	N/A	82130142	9305522
sha	N/A	N/A	29232741	6494310
susan	N/A	N/A	28564838	53017

Benchmarks. We use the Mibench2 benchmark suite [18]. Mibench2 (or parts thereof) is used in a vast fraction of intermittent computing literature [2–4, 7, 13, 20, 22, 25, 30]. We are particularly interested in memory operations. Tab. 1 shows the number of memory access operations of each benchmark in Mibench2, using either MSP430 or ARM Cortex M* processor architectures. The numbers provide evidence of the diversity among different benchmarks.

Note that, as in existing literature [22], we consider only five benchmarks on the MSP430 architecture because the memory footprints of the other benchmarks are excessive compared to the data memory of the MSP430.

Oracle. As a baseline to compare against, we use Gem5 or MSPSim to trace the execution of all feasible benchmarks and examine all memory operations afterwards. Based on this information, we identify *each and every* (temporary) silent store, simply by looking at what data the program writes at what memory address, and when.

The measures we collect this way effectively represent an ORACLE that knows exactly what (temporary) silent stores occur when, because it relies on post-facto complete knowledge on the program execution. This performance is unattainable in practice, but is useful as a yardstick to understand how practical solutions compare with the theoretical optimum.

4 On the Number of Silent Stores

We investigate whether (temporary) silent stores are found in the benchmarks we consider, using the two aforementioned processor architectures. Verifying their existence is a necessary stepping stone to accurately identify the source of energy gains they possibly enable, or the reason for their absence.

We split the discussion between what we observe with the ORACLE, that is, a technique based on post-processing detailed program traces, yet inapplicable in reality, and how a dedicated store buffer may perform at run-time in real executions.

4.1 Theory → Oracle

Fig. 3 shows the results on the number of (temporary) silent stores from the ORACLE. The chart demonstrates that, on average, about 40% of all store instructions are silent, and another 5% are temporary silent. The CRC benchmark appears as an

Figure 3. Percentage of (temporary) silent stores in Mibench 2 on MSP430 and ARM Cortex M* architectures, obtained with the ORACLE. *On average, 40% (5%) of the total number of store instructions are (temporary) silent.*

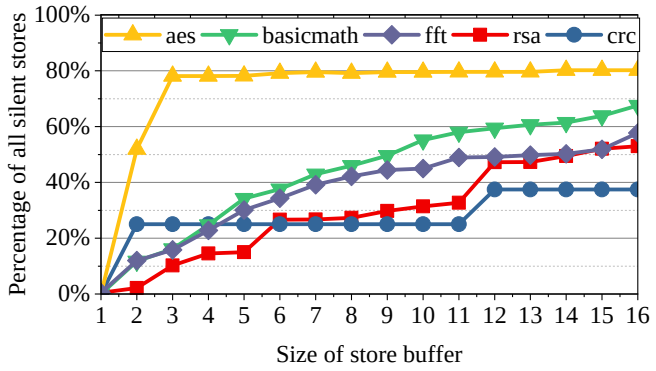

Figure 4. Percentage of all silent stores captured using a variable size store buffer on an MSP430 processor architecture. *The ability of the store buffer to detect silent stores increases with its size. On average, a 16-entry store buffer can capture over 59% of all silent stores.*

outlier on both processor architectures. This is because CRC is a compute-intensive benchmark and, in the worst case, every bit of value in a CRC needs to be explicitly computed.

We observe that the number of silent stores is much higher than the number of temporary silent stores. We conjecture that this is because of the limited number of internal registers in modern embedded processors. For example, there are only 12 general-purpose registers in MSP430RF* architectures and 12 to 31 general-purpose registers in ARM Cortex M* architectures. When the register file is full and a load instruction executes without an available register, the existing value in the register needs to write back to memory. This form of register spilling causes a silent store without any explicit store instruction, which increases the number of silent stores overall.

The abundance of silent stores we observe prompts us to further our study and better understand *i)* how many of these silent stores may be practically captured at run-time, which we examine next, and *ii)* what are the energy savings stemming from eliminating silent stores, which we investigate in Sec. 5.

4.2 Practice → Store Buffer

Fig. 4 shows the percentage of silent stores, out of all available silent stores in a program, that can be practically captured using a store buffer of different size on an MSP430 architecture. A store buffer can capture over 75% of all silent stores in AES and 38% on average when it has 4 entries. When the store

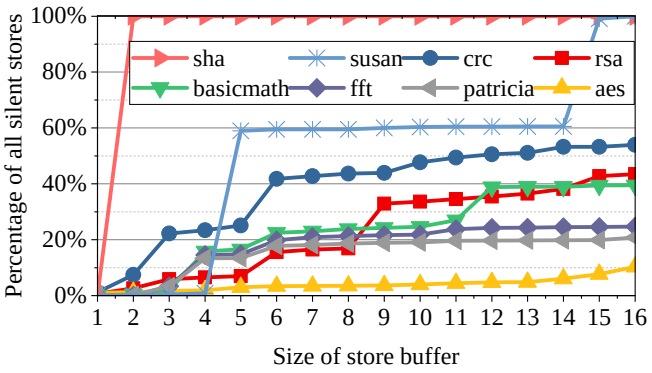

Figure 5. Percentage of all silent stores captured using a variable size store buffer on an ARM Cortex M* processor architecture. *On average, the store buffer captures up to 42.9% of all silent stores when it has 16 entries. However, the results show significant variance across different benchmarks.*

buffer has 16 entries, it can, on average, capture over 59% of all silent stores. With an MSP430 architecture, a store buffer appears reasonably effective at capturing silent stores.

Fig. 5 demonstrates the ability of a store buffer to capture silent stores on ARM Cortex M* processor architectures. On average, the store buffer captures up to 42.9% of all silent stores when it has 16 entries. In SHA, all silent stores potentially existing in the code may be captured already with only two entries in the store buffer; in AES, this figure drops to only 6.7% of all silent stores, even with 16 entries in the store buffer. Since the experimental results show great variability, we conclude that the efficiency of store buffers in ARM Cortex M* architectures is dependent on the characteristics of benchmarks.

5 On the Energy Gains of Silent Stores

Building upon the results of Sec. 4, we proceed with measuring the potential energy savings enabled by silent stores. This time, our discussion is three-pronged.

First, we analytically compute the energy saving enabled by removing *from machine code* all silent stores that the ORACLE identifies, as per the results of Fig. 3. This entails that the device spares not just the energy of the actual write to memory, but also the cost of instruction fetch and execution. The figures we obtain this way represent a theoretical upper bound, yet unattainable in practice. Next, we measure the energy gains in a device with an *infinite-size* store buffer. This provides an intermediate point for performance evaluation: it is also unattainable in practice, but represents a step towards an actual run-time solution and it does account for instruction fetch and execution. Most importantly, the infinite-size store buffer serves to understand the gap between a run-time upper bound and what can be practically achieved with a *finite-size store buffer*, whose ability to capture silent stores is inherently limited and whose performance we also measure.

5.1 Theory → Oracle

Fig. 6 shows the energy savings that could be achieved by removing the silent stores identified by the ORACLE from the machine code. On average, the ORACLE can save 7.0% of the total energy consumption on an MSP430 architecture; whereas it can save 8.2% of the same figure on an ARM Cortex M* architecture. The highest energy savings is achieved when running SHA. CRC and susan show lower energy savings than

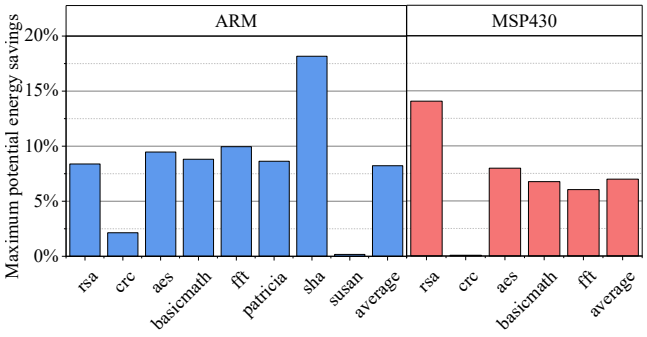

Figure 6. Energy savings obtained by removing from machine code all silent stores the ORACLE identifies. *On average, the energy savings top at 7.0% and 8.2% on MSP430 and ARM Cortex M* architectures, respectively.*

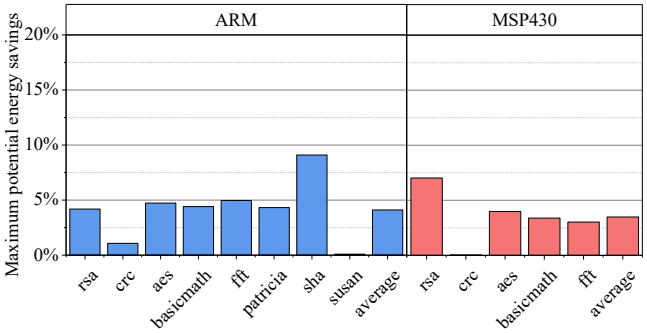

Figure 7. Energy savings obtained by running the benchmarks with an infinite-size store buffer. *The energy savings are only 3.5% and 4.1% on the MSP430 and ARM Cortex M* architectures, respectively.*

other benchmarks because susan is a memory-read-intensive benchmark with limited register spilling, as Tab. 1 shows; CRC includes a low number of silent stores, as per Fig. 3.

5.2 Theory → Infinite-size Store Buffer

Fig. 7 shows the energy savings obtained by running with an infinite-size store buffer. On average, the MSP430 architecture offers a 3.5% energy gain, whereas the ARM Cortex M* shows a 4.1% energy gain.

Note that these numbers *do not* account for the energy overhead of the store buffer itself, and yet they are already halved compared to Fig. 6. This is expected for two reasons. The infinite-size store buffer is structurally unlimited, so its ability to capture silent stores eventually equals that of the ORACLE, even though it operates at run-time rather than offline. For example, the store buffer in an MSP430 architecture that can capture all silent stores for fft has 7776658 entries. The overhead of such large store buffer is unacceptable in battery-less IoT devices. However, by operating at run-time, this configuration still pays the energy cost of instruction fetch and execution, while only sparing the cost of the actual memory write.

5.3 Practice → Finite-size Store Buffer

We combine the results of Fig. 4 and Fig. 5 with the energy figures of Sec. 3 and compute the energy gains that a variable size store buffer may offer.

Fig. 8 shows the results for the MSP430 architecture. The chart indicates that the energy savings in a practical configuration are ultimately minimal: on average, a mere 2.3% of the total energy consumption with a 16-entry store buffer. Fig. 9

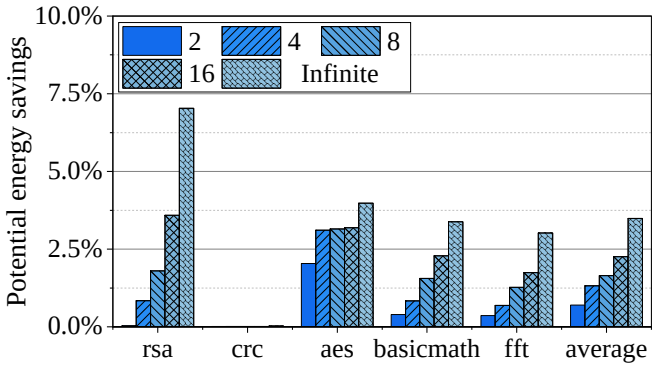

Figure 8. Energy savings obtained by running the benchmarks with a variable size store buffer on an MSP430 architecture. *With a finite-size store buffer, energy savings are further limited. On average, only a 2.3% improvement can be achieved with a 16-entry store buffer.*

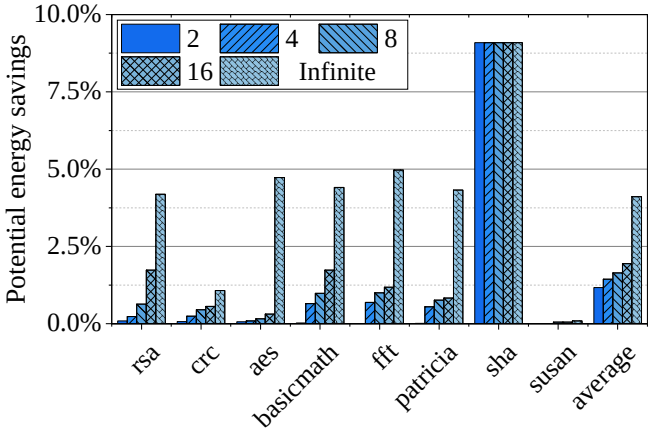

Figure 9. Energy savings obtained by running the benchmarks with a variable size store buffer on an ARM Cortex M* architecture. *On average, the 16-entry store buffer saves only 1.9% of the total energy, leading to the same conclusion as for the MSP430 architecture.*

shows the results for the ARM Cortex M* architecture. On average, only 1.9% of the energy may be saved with a 16-entry store buffer, hence leading to the same conclusion.

Interestingly, Fig. 8 shows that only four entries in the store buffer suffice in the AES benchmark to save as much energy as a 16-entry store buffer, and both are close to the performance of an infinite-size store buffer. For RSA, the trends are markedly different: the 16-entry store buffer performs much better than any other configuration, but is still far from the performance of an infinite-size store buffer. This is coherent with the results of Fig. 4, where AES is the only benchmark where a practical store buffer can capture most silent stores.

Fig. 9 similarly shows that in SHA, the energy savings with a 2-entry store buffer match the ones of an infinite-size store buffer, as two entries are enough to capture all silent stores, as shown in Fig. 5. However, the energy savings observed in other benchmarks are quite far from the ones enabled by an infinite-size store buffer, which is consistent with the results of Fig. 5. The latter figure shows indeed that not even 16 entries in the store buffer suffice to capture most silent stores.

The difference in performance across benchmarks provides an interesting perspective on how diverse is the workload of

battery-less IoT devices. Using the MSP430 architecture, programs exist, such as AES, that are structurally designed in ways that store instructions are sufficiently close to each other that a small store buffer is sufficient to capture most of the silent stores. Programs such as RSA, instead, do not show this characteristic and would require large store buffers to do the same.

6 Discussion

Our work certainly has limitations. We may, for example, investigate new compilation techniques that use different memory layouts or register spilling techniques that increase the availability of silent stores. Not considering the complexity of designing these techniques, the results we obtain with the ORACLE determine the upper bound. This is arguably not sufficiently large to justify the massive efforts required for conceiving a dedicated compilation pipeline.

We attribute the negative results we obtain to a fundamental difference in the key performance metrics for battery-less IoT devices, compared to general-purpose computing systems where silent stores offer significant advantages. Indeed, the quest for energy efficiency over any other performance improvement [12] has two consequences.

1. Most existing techniques to capture silent stores are inapplicable due to the tremendous energy overhead they would cause, as we argue in Sec. 2. The applicable techniques, however, show severe limitations in their ability to capture silent stores, as we report in Sec. 4.
2. The energy performance, word-level addressing, and symmetry in the energy cost of read and write operations of modern NVM technology, such as FRAM, drastically close the gap to the volatile counterpart. Saving memory operations on FRAM, therefore, no longer represents such a huge energy saving, as shown in Sec. 5.

Beyond specifically investigating the applicability of silent stores in intermittent computing, the key contribution of this paper is to inform future efforts. Should energy consumption remain the chief performance metric, FRAM technology arguably places upcoming research in a situation where energy savings are to be sought elsewhere.

7 Conclusion

We investigated the applicability of silent stores in battery-less IoT devices that compute intermittently. We demonstrated, using common IoT architectures, that silent stores abound in representative benchmarks, yet the energy improvements they enable is unfortunately minimal. Resource constraints and the features of modern NVM technology, such as FRAM, indeed largely neutralize the potential benefits stemming from capturing silent stores.

Acknowledgements. This work is supported by the Swedish Foundation for Strategic Research (SSF).

8 References

[1] M. Afanasov et al. Battery-less zero-maintenance embedded sensing at the mithræum of circus maximus. In *International Conference on Embedded Networked Sensor Systems (SENSYS)*, 2020.

[2] S. Ahmed et al. Efficient intermittent computing with differential checkpointing. In *International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2019.

[3] D. Balsamo et al. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters (ESL)*, 2014.

[4] D. Balsamo et al. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2016.

[5] N. Bhatti et al. Energy harvesting and wireless transfer in sensor network applications: Concepts and experiences. *ACM Transactions on Sensor Networks (TOSN)*, 2016.

[6] N. Bhatti and L. Mottola. Efficient state retention for transiently-powered embedded sensing. In *International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2016.

[7] N. Bhatti and L. Mottola. Harvos: Efficient code instrumentation for transiently-powered embedded sensing. In *International Conference on Information Processing in Sensor Networks (IPSN)*, 2017.

[8] N. Binkert et al. The gem5 simulator. *ACM SIGARCH computer architecture news*, 2011.

[9] A. Colin and B. Lucia. Chain: tasks and channels for reliable intermittent programs. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2016.

[10] J. Eriksson et al. COOJA/MSPSim: interoperability testing for wireless sensor networks. In *International Conference on Simulation Tools and Techniques (SIMUTools)*, 2009.

[11] J. Hennessy and D. Patterson. *Computer Architecture A Quantitative Approach*. 2017.

[12] J. Hester and J. Sorber. The future of sensing is batteryless, intermittent, and awesome. In *International Conference on Embedded Networked Sensor Systems (SENSYS)*, 2017.

[13] M. Hicks. Clank: Architectural support for intermittent computation. *ACM SIGARCH Computer Architecture News*, 2017.

[14] P. Krishnamurthy et al. Evaluating dusty caches on general workloads. In *International Workshop on Duplicating, Deconstructing, and Debunking (WDDD)*, 2006.

[15] K. Lepak and M. Lipasti. Silent stores for free. In *International Symposium on Microarchitectures (MICRO)*, 2000.

[16] K. Lepak and M. Lipasti. Temporally silent stores. *ACM SIGARCH Computer Architecture News*, 2002.

[17] B. Lucia and B. Ransford. A simpler, safer programming and execution model for intermittent systems. *ACM SIGPLAN Notices*, 2015.

[18] M. Hicks. Mibench2. `https://github.com/impedimentToProgress/MiBench2`.

[19] K. Maeng, A. Colin, and B. Lucia. Alpaca: Intermittent execution without checkpoints. *Internatonal Conference on Programming Languages (PACMPL)*, 2017.

[20] K. Maeng and B. Lucia. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *International Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[21] A. Maioli et al. Discovering the hidden anomalies of intermittent computing. In *International Conference on Embedded Wireless Systems and Networks (EWSN)*, 2021.

[22] A. Maioli and L. Mottola. Alfred: Virtual memory for intermittent computing. In *International Conference on Embedded Networked Sensor Systems (SENSYS)*, 2021.

[23] A. Majid et al. Dynamic task-based intermittent execution for energy-harvesting devices. *ACM Transactions on Sensor Networks (TOSN)*, 2020.

[24] F. Pereira, G. Leobas, and A. Gamatié. Static prediction of silent stores. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2018.

[25] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on rfid-scale devices. In *International Conference on Architectural support for Programming Languages and Operating systems (ASPLOS)*, 2011.

[26] J. Shen and M. Lipasti. *Modern Processor Design*. 2005.

[27] D. Stahlberg, F. Eller, A. Maass, and D. Frey. We knew it all along: Hindsight bias. *Organizational Behavior and Human Decision Processes*, 63(1):46–58, 1995.

[28] A. Suresh, P. Cicotti, and L. Carrington. Evaluation of emerging memory technologies for hpc, data intensive applications. In *International Conference on Cluster Computing (CLUSTER)*, 2014.

[29] Texas Instruments. MSP430FR5969 Data Sheet. `www.ti.com/document-viewer/MSP430FR5969/datasheet`, 2012.

[30] J. Woude and M. Hicks. Intermittent computation without hardware support or programmer intervention. In *USENIX Symposium on Operating Systems Design and Implementation (ODSI)*, 2016.

[31] K. Yildirim et al. Ink: Reactive kernel for tiny batteryless sensors. In *International Conference on Embedded Networked Sensor Systems (SENSYS)*, 2018.