Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation

Alberto Rodriguez†, Domenico Balsamo†, Anup Das†,
Alex S. Weddell†, Davide Brunelli‡, Bashir M. Al-Hashimi†, Geoff V. Merrett†
†Department of ECS, University of Southampton
‡Department of Electronics, University of Trento
†{ara1g13, db2a12, a.k.das, asw, bmah, gvm}@ecs.soton.ac.uk,
‡davide.brunelli@unitn.it

ABSTRACT

Systems operating from harvested sources typically integrate batteries or supercapacitors to smooth out rapid changes in harvester output. However, such energy storage devices require time for charging and increase the size, mass and cost of the system. A recent approach to address this is to power systems directly from the harvester output, termed transient computing. To solve the problem of having to restart computation from the start due to power-cycles, a number of techniques have been proposed to deal with transient power sources. In this paper, we quantitatively evaluate three state-of-the-art approaches on a Texas Instruments MSP430 microcontroller characterizing the application scenarios where each performs best. Finally, recommendations are provided to system designers for selecting the most suitable approach.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Transient Computing, Energy Harvesting, Wind Turbines, Photo Voltaic Cells

Keywords

Checkpoint, Hibernus, IoT, Mementos, QuickRecall

1. INTRODUCTION

The Internet-of-Things (IoT) is the interconnection of billions of things. Each IoT device could be considered as an ultra-low power and resource-constrained sensor elaboration platform. Power management of these devices is emerging as a primary challenge for system designers as they typically have to last for few years without intervention to charge or replace batteries. Energy harvesting (EH) is an efficient solution to power sensor nodes present in these connected devices. EH sources harvest electric power from ambient sources or human motion including light, vibration, or temperature differences [2, 8, 11]. This harvested energy is converted into a DC signal (voltage and current) that is used to power up sensor devices. EH power sources are typically intermittent due to temporal variation in the environmental parameter (e.g., time of day, weather condition and available light). 1 shows an example of the harvested power from a micro-wind turbine and a photovoltaic module. The voltage of the micro-wind turbine varies between -4V to 4V. The frequency of the power-cycle is dependent on the wind velocity. Similarly, the output from the photovoltaic cell, used in this example, changes depending on the intensity of the light source. An application executed on the sensor node powered by these EH sources can potentially be interrupted depending on the harvested power availability. To overcome this limitation, sensor nodes typically integrate energy storage in the form of supercapacitors to buffer energy in order to sustain computation at times of power unavailability.

Energy storage devices require time to power on and increases system size, mass and cost. An alternative solution is to power a system directly using the harvested source, eliminating the need of an energy storage device. However, to sustain computation with transient sources, the common approach is to checkpoint the system state to save it into a non-

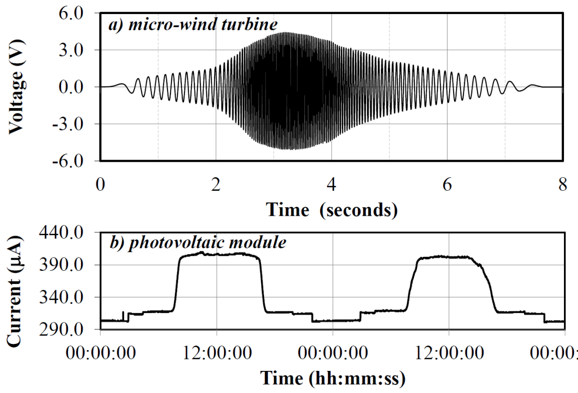

Figure 1: Harvester output from (a) micro-wind turbine and (b) photovoltaic module

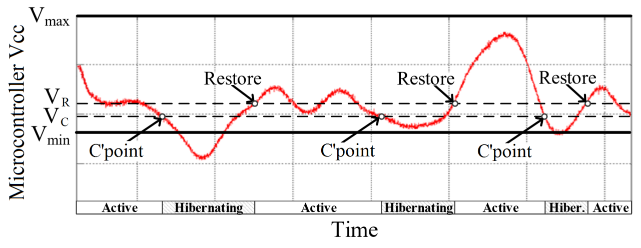

Figure 2: Checkpointing and restore to sustain operation with transient energy harvester outputs.

volatile memory. Later when the supply is restored, the last checkpoint saved before power loss is restored and the operation is continued from the point where it was halted. Three prominent techniques – *Mementos* [10], *QuickRecall* [5] and *Hibernus* [1] have been proposed based on the checkpointing concept, differing in the checkpointing approach leading to different timing/power overheads. This work studies these three approaches and evaluates them theoretically and experimentally on a common platform – the Texas Instruments MSP430FR microcontroller. The objective is to identify scenarios where one approach outperforms others.

The remainder of this paper is organized as follows. A brief background on the related works is provided in Section 2. This is followed by a description of the three approaches in Section 3. The quantitative evaluation of these approaches is provided in Section 4. Finally, the paper is concluded in Section 5 with recommendations for system designers.

2. RELATED WORKS

A new paradigm, which addresses the presented challenges, is of 'transiently-powered computing' [7] allowing systems to operate reliably from intermittent or limited sources such as energy harvesting. This borrows from the concept of checkpointing, which has been used in large-scale computing for decades to provide robustness against errors or hardware failure [3]. This technique involves systematically saving data to non-volatile memory (NVM). To recover from a failure, systems roll back to the previous valid checkpoint, before continuing operation. State-of-art embedded systems use a variety of classic and advanced NVM structures to save their state. Examples of memories used for state retention are Flash or battery-backed SRAM memories [6]. However, a drawback of checkpointing is that it is impossible to predict the exact time of failures, so computation time and energy will be wasted by (1) taking unnecessary checkpoints, and (2) rolling back by the period between the checkpoint and failure. Attempts have been made to address these problems, for example by assuming different failure distributions. Moreover, system shut-down and wake-up have significant time and energy cost, and they must be minimized.

Recently, the checkpointing concept has been applied to embedded devices with unstable power supplies, to avoid power-cycling causing the loss of data and to enable long-running computations across several power cycles. This is enabled by systems saving their state so that, when their power supply fails, they can resume operation when it recovers. Figure 2 shows the output voltage of an energy harvester, which is used to power a microcontroller. Whenever, the supply voltage crosses the checkpoint threshold V_c, a checkpoint is stored in the NVM. On the other hand, whenever the supply voltage crosses the restore threshold V_R, a checkpoint is restored. As shown in Figure 2, this allows computation to continue across several power-cycles, which would conventionally have caused a system to reset repeatedly. A few recently published papers show that the time and energy cost of distributed state-retentive logic elements can be lowered by orders of magnitude with respect to traditional Flash-based approaches using alternative non-volatile memory technology, such as FRAM [4]. Some more advanced technologies are currently under development, such as ReRAM [9], which could further reduce NVM storage energy and cost.

Prominent works in this area include (1) Mementos [10], which uses checkpoints placed at compile-time to save periodic snapshots of system state to NVM; (2) Hibernus [1], which monitors the external voltage to store RAM and register contents in NVM when a power failure is imminent and (3) QuickRecall [5], a refinement of Hibernus, which uses NVM as a a unified memory; and these three techniques are discussed in more details in the subsequent section.

3. TRANSIENT COMPUTING METHODS

In this section, we describe the three transient computing techniques in details.

3.1 Mementos

The first presented solution is Mementos [10], which uses checkpoints placed at a compile-time. It saves periodic snapshots of system state to non-volatile memory (NVM), which enables it to return to a previous checkpoint after a power failure. Mementos uses the following three different heuristics to insert checkpoints and verify the input voltage level.

- The first heuristic is the `loop-latch mode`. Here, Mementos inserts a trigger point for every loop of the program in order to check the input voltage level at each iteration.

- The second heuristic is the `function-return mode`. In this mode, Mementos inserts trigger points after every function call in order to check the input voltage level when the program returns from a function call.

- The third heuristic is the `timer-aided mode`. This heuristic works in conjunction with the two previous heuristics. Here, Mementos inserts a timer interrupt that sets a flag at predefined execution intervals. At the trigger points, the voltage level is checked only if the flag is set. This heuristic avoids frequent checkpointing, saving energy. We will not consider in the quantitative evaluation section this last technique.

Mementos also has the option of inserting trigger points manually in any position of the program or forcing a snapshot without checking the input voltage level. In order to predict a possible power failure, Mementos compares the input voltage against a threshold by using an analog-to-digital converter (ADC). For Mementos, the checkpoint threshold can be calculated considering a constant current draw I so that the time Δt between two voltage levels V and V_{min} is $\Delta t = C\ (V\text{-}V_{min})/I$. However, a factor complicates the task of checking: Mementos's ability to precisely complete a checkpoint depends on the frequency of trigger points.

This is the main reason why we decided to fix V_{min} bigger than necessary ($V_{min} = 2.4V$), assuming that no energy will be harvested between a trigger point and a power failure. When the supply voltage reaches this threshold, the system considers an imminent power failure and starts checkpointing. Mementos uses two memory blocks and alternates between saving to each of them in order to have always a state-saved start. When power is available again, Mementos looks for a valid checkpoint and copies its content into RAM and registers to continue the program execution from the point it was stopped. The main application of Mementos is on RFID-scale devices powered by a RF-harvesting source, which stores the energy in a capacitor.

Disadvantages of this approach include the fact that many checkpoints will be taken (most of which will be redundant) and that space must be reserved in non-volatile memory for two complete checkpoints in case a power interruption occurs whilst one is being taken.

3.2 Hibernus

Hibernus is the second presented solution, a refinement to Mementos technique for sustaining computation powered by intermittent sources. Hibernus stores a snapshot before a power failure without inserting trigger points in the main program [1]. This technique allows to save only one snapshot every power failure. Hibernus has two states: Active, when the input voltage level is over a restore value (V_R). Hibernating when input voltage is below a threshold (V_H).

Hibernus is implemented on a TI MSP430FR5739 microcontroller which has an internal comparator that was used to send an interruption when the input voltage level crosses either hibernate or restore thresholds. This method uses the FRAM as a non-volatile memory. In order to save a snapshot, Hibernus uses the energy stored in the decoupling capacitance of the microcontroller. This allows to have a low V_H value which increases the active period of the main program. V_H is determined considering the time required to charge the decoupling capacitor in order to have enough energy for saving a snapshot before a power failure. In order to obtain the threshold value, first, it was calculated the energy required to save a snapshot, which is obtained as follows [1]:

$$E_\sigma = n_\alpha E_\alpha + n_\beta E_\beta \quad (1)$$

Where n_α is the number of bytes of the RAM, n_β the number of bytes used by registers, E_α and E_β are the energy required to copy RAM contents and the registers respectively. The microcontroller works in a range of voltage between V_{min} and V_{max}. Given the total capacitance ($\sum C$), the energy E_δ stored in the decoupling capacitor between a given voltage V and V_{min} is calculated as follows [1]:

$$E_\delta = \frac{V^2 - V_{min}^2}{2}.\sum C \quad (2)$$

Inspecting the parameters of the microcontroller [4], it was obtained that the total capacitance is 16μF, and the size in bytes of the RAM and core registers is 1024 and 512 bytes respectively. 4.2nJ energy is needed to save a byte into RAM (E_α) and 2.7nJ in case of FRAM (E_β). Substituting these values in (1) it is obtained that to save a snapshot consumes 5.7 μJ (E_σ). To save a complete snapshot requires that $E_\sigma \leq E_\delta$. The microcontroller works in a range from V_{min}=1.9V to V_{max}=3.6V and the obtained threshold, considering $E_\sigma = E_\delta$, to save a complete snapshot is 2.17V. In order to add hysteresis, V_R was set higher to allow V_{cc} to be over V_H. An internal comparator is checking the voltage level and when it is below V_H, the comparator generates an interrupt. Inside the interrupt handler a function is called to save the snapshot into FRAM, the checkpointing is set and then, system enters in low-power mode. Whether the input voltage is never lower than 1.9V and its value rises again over V_R, the system exits from low-power mode and continues where it was stopped without restoring the whole system. In the case that the input voltage goes below 1.9V, the microcontroller is turned off. When the energy is available again, the system first checks the flag. If the flag is set means that a snapshot is saved. Therefore, Hibernus restores the RAM's contents and the registers' values. Then, it resets the flag and the program continues where it was interrupted. Hibernus is transparent to the programmer. It just need to include *hibernus.h* file that contains all the functionality and call the routines *initialise()*, *hibernate()* and *restore()*.

3.3 QuickRecall

The last proposed solution is QuickRecall [5], which is similar to Hibernus but it allows FRAM to be also utilized as RAM, enabling the system to work as an "unified memory system". In this way, only the FRAM is used as a unified memory while the system's RAM is not used. In order to check the voltage level, the system uses an external comparator, which is connected to the GPIO pins of the microcontroller. This comparator is configured with a trigger voltage (V_{trig}) and sends a signal output when the input voltage level (V_{cc}) is smaller than V_{trig}. The value of trigger voltage is not required to be relative high, unlike Mementos, because it just needs to back up peripherals, program counter, stack pointer, status register and general purpose registers (GPR) before a power failure occurs. Thus, it requires a value of 2.0003V. QuickRecall uses a flag which is set during checkpointing. This flag is used by system to know whether there is a stored checkpoint or not after a power failure. If flag is set, all peripherals are initialized; a check is performed to determine if $V_{cc} > V_{trig}$: if so, core registers are restored, the flag is cleared and the main program is executed. A possible disadvantage of QuickRecall is that it relies on the use of a processor with a unified FRAM memory.

4. QUANTITATIVE EVALUATION

In this section we first evaluate the three techniques mathematically, establishing the scenario where one technique outperforms the others. Later we validate the same using a signal generator on a common microcontroller platform.

4.1 Mathematical Evaluation

4.1.1 Execution Time Comparison

The total time, T_{hibernus}, to execute a test algorithm with *Hibernus* is given by (3), where T_a is the CPU time required to execute the algorithm, n_ι is the number of power interruptions (where $V_{\text{cc}} < V_{\text{min}}$) per algorithm execution, T_s is the time required to save a snapshot to NVM, T_r is the time required to restore from NVM memory, and $\overline{T_\lambda}$ is the average time spent sleeping (after a snapshot has been saved but before $V_{\text{cc}} = V_{\text{min}}$, and on power-up when $V_{\text{min}} < V_{\text{cc}} < V_{\text{R}}$). The absolute limit of supply interruption frequency, f_ι, is $1/(T_s+T_r)$. The execution time of the QuickRecall is similar to that of the Hibernus and is therefore given by Equation 3.

$$\underbrace{T_{\text{Hibernus\_QuickRecall}}}_{\text{Total execution}} = \overbrace{T_a}^{\text{Algorithm}} + \underbrace{n_\iota}_{\text{No. interruptions}} (\overbrace{T_s}^{\text{Save snapshot}} + \underbrace{T_r}_{\text{Restore snapshot}} + \overbrace{\overline{T_\lambda}}^{\text{Sleep}}) \quad (3)$$

The total time, T_{mementos}, to execute an algorithm with Mementos is given by (4), where n_m is the number of checkpoints per complete execution of the algorithm, T_m is the time taken for an ADC reading of V_{cc}, and ρ_s is the proportion of checkpoints resulting in a snapshot, taking T_s.

$$\underbrace{T_{\text{mementos}}}_{\text{Total execution}} = \overbrace{T_a}^{\text{Algorithm}} + \underbrace{n_\iota}_{\text{No. interruptions}} (\overbrace{T_r}^{\text{Restore snapshot}} + \underbrace{\frac{T_a}{2n_m}}_{\text{Backtrack}}) + \overbrace{n_m(T_m + \rho_s T_s)}^{\text{Monitoring and save snapshot}} \quad (4)$$

Hence, $T_{\text{hibernus}} < T_{\text{mementos}}$ provided $n_\iota(T_a/2n_m) + n_m T_m + (n_m\rho_s - n_\iota)T_s > n_\iota \overline{T_\lambda}$; that is, *Hibernus* spends less time sleeping than Mementos spends on backtracks (re-running code that was executed between a snapshot and a power interruption), sampling V_{cc}, and redundant snapshot saves. This is evaluated experimentally in the next section.

4.1.2 *Comparison of Energy Consumption*

Let P_F and P_R denote the average power consumption for accessing the FRAM and RAM, respectively. Usually, $P_F > P_R$. In QuickRecall, both the application code and dynamic data structures are stored in FRAM, while in Hibernus, the application code resides in the FRAM while the dynamic data structures in the RAM.

When the system is powered using a time varying source, e.g., a sinusoidal signal, both Hibernus and QuickRecall approaches behave similarly by storing and restoring checkpoints. The energy overhead for checkpoints in the two approaches can be evaluated as follows.

The energy consumed by Hibernus, $E_{hibernus}$, depends on the size of the volatile memory and the energy consumption for copying each byte.

$$E_{hibernus} = n_\alpha E_\alpha + n_\beta E_\beta \quad (5)$$

Here, n_α and n_β are the sizes of the RAM and registers (in bytes) respectively. E_α and E_β are the energy required to copy each RAM and register byte to NVM (J/byte).

The energy consumed by QuickRecall is given by

$$E_{quickrecall} = n_\beta E_\beta \quad (6)$$

Clearly, $E_{quickrecall} < E_{hibernus}$. As can be seen, the energy for checkpointing and restore for QuickRecall is lower than that of Hibernus. However, for a system powered by a DC source, the energy consumption of Hibernus is lower than that of QuickRecall. We are interested in finding the crossover frequency where one technique outperforms the other. To do so, it is important to note that, in each power cycle, the system will hibernate and restore once. Assuming f is the frequency of input source, the energy overhead of checkpointing and restore for Hibernus = $E_{hibernus} - E_{quickrecall}$. The crossover frequency is given by

$$f\_cross = \frac{(P_F - P_R)}{(E_{hibernus} - E_{quickrecall})} \quad (7)$$

It is important to note that P_R and P_F depend on the application code and hence the crossover frequency is dependent on the application being executed. This crossover frequency is validated experimentally in the next section.

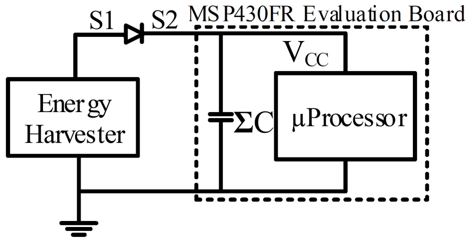

Figure 3: Experimental setup

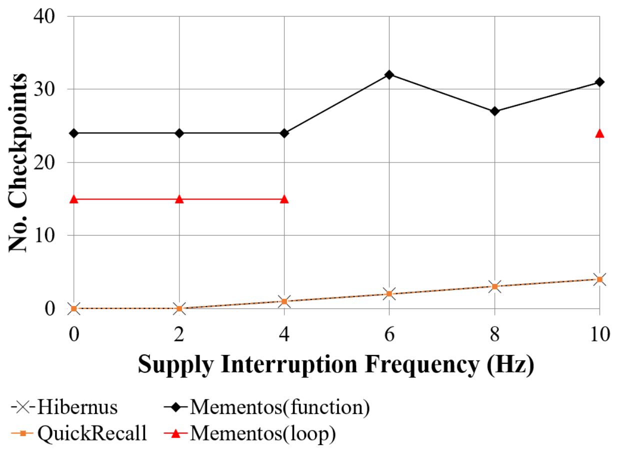

Figure 4: Number of checkpoints of the three approaches.

4.2 Experimental Validation

4.2.1 *Experimental Setup*

This section provides the experimental validation of the three approaches implemented on a TI MSP430FR5739 microcontroller. This platform has 1KB of RAM and 16KB of FRAM. To perform the required experiments, a signal generator is used to power the system, and a DC power analyzer is used to record power consumption. Figure 3 plots the experimental setup used for Hibernus, QuickRecall and Mementos. As shown in this figure, the microcontroller is powered using an energy harvester through the diode. C represents the total on-board decoupling capacitance. The internal voltage comparator of the MSP430FR platform is used for voltage comparison for all the three approaches.

4.2.2 *Application Scenarios*

The microcontroller's clock is configured to run at 8MHz executing the FFT application, which analyses three arrays, each holding 128 8-bit samples of tri-axial accelerometer data. The system is powered with two different sources – a 3.4V DC and Sinusoidal sources with ±3.4V amplitude operating at frequencies ranging from 2 Hz to 10 Hz.

4.2.3 *Number of Checkpoints Executed*

Figure 4 shows the number of checkpoints executed by the three transient computing approaches during the execution of the FFT. A range of supply frequencies (2-10 Hz, and

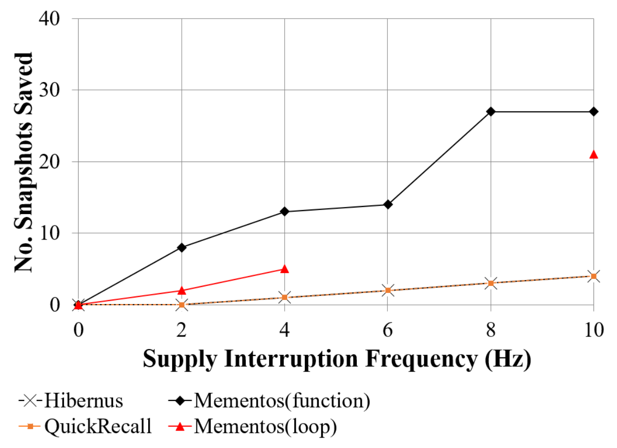

Figure 5: Number of snapshots of the three approaches.

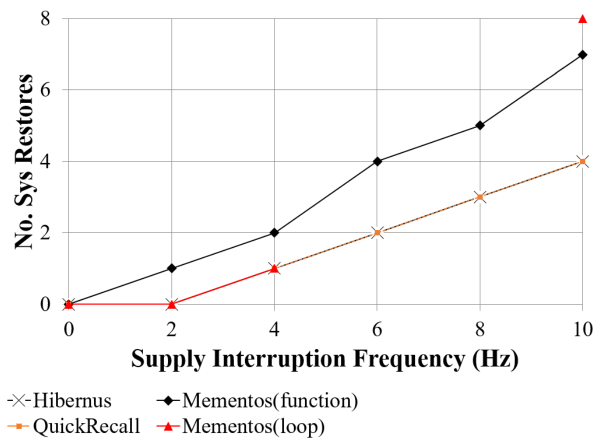

Figure 6: Number of restores of the three approaches.

DC) were chosen to represent the intermittent power output that may be expected from a high-power EH. As can be seen, Hibernus and QuickRecall modulate the number of times snapshots are taken as a function of the supply interruption frequency, while Mementos executes a static number of checkpoints (15 and 24 times), although some are repeated when $V_{cc} < V_{min}$ during a snapshot. Moreover, Mementos (loop approach) operates unstably with frequencies higher than 4 Hz due to the static and uneven placement of checkpoints at compile time: checkpoints are only inserted at function calls or loops. In cases where the supply is interrupted in the period between a restore and the next snapshot being saved, the system can become 'stuck', i.e. executes the same portion of code from the last saved checkpoint before $V_{cc} < V_{min}$ without reaching or being able to save a snapshot at the next checkpoint.

4.2.4 *Number of Snapshots Executed*

Figure 5 shows the number of snapshots saved by the three approaches. Hibernus and QuickRecall saves a snapshot every time the hibernate routine is executed, while Mementos saves a snapshot only when $V_{cc} < V_{min}$. The number of snapshots with Mementos is therefore correlated to each checkpoint placement, the value of V_{min} and the supply interruption frequency, while for Hibernus and QuickRecall this depends on the supply interruption frequency only.

4.2.5 *Number of Restores Executed*

Fig. 6 shows that Hibernus and QuickRecall complete execution of the FFT application over the same number of power interruptions while Mementos takes for both loop and function approaches a bigger number of cycles.

4.2.6 *Time Overhead*

Figure 7 plots the time overhead of the three approaches for different interruption frequencies while executing the FFT application. As established mathematically earlier in this section, the time overhead of Mementos is much higher than that of QuickRecall and Hibernus. This is also validated in the figure. It is important to observe that as the supply interruption frequency increases, the execution time overhead of Mementos in the function mode increases rapidly, increasing to over 100% overhead (2x execution time) for an

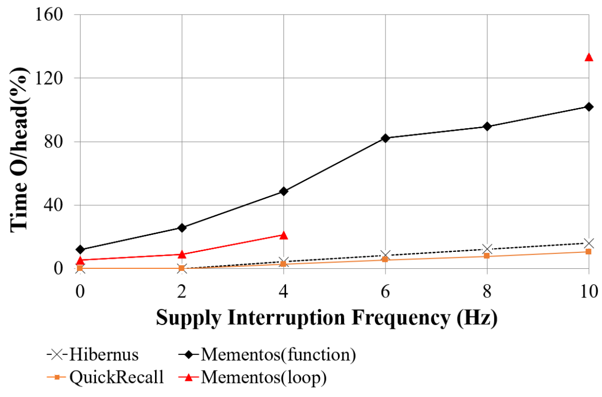

Figure 7: Time Overhead of the three approaches.

interruption frequency of 10 Hz. Finally, the time overhead for QuickRecall is similar to that of the Hibernus approach.

4.2.7 *Current Consumption*

Figure 8 reports the current consumption of QuickRecall and Hibernus using a low-frequency input source while executing the FFT application. The current peaks in the figure correspond to the time when the microcontroller is on. At other times, the current is very close to zero. This is because at these times, the microcontroller is in hibernate state and does not consume any current. It is important to note that the current consumption of Mementos is similar to that of Hibernus and is therefore not included. As can be seen from the figure, the current consumption of QuickRecall is higher than that of Hibernus.

4.2.8 *Hibernus vs QuickRecall as a function of Interruption Frequencies*

Figure 9 plots the energy results for QuickRecall and Hibernus as a function of the supply interruption frequency while executing the FFT application. The system is powered using a square wave generator to simulate the interruption behavior. An interruption frequency f signifies that the system is interrupted f times per second. In other words, the system is interrupted every $1/f$ seconds. The interruption frequency reported in the figure covers the typical scenarios

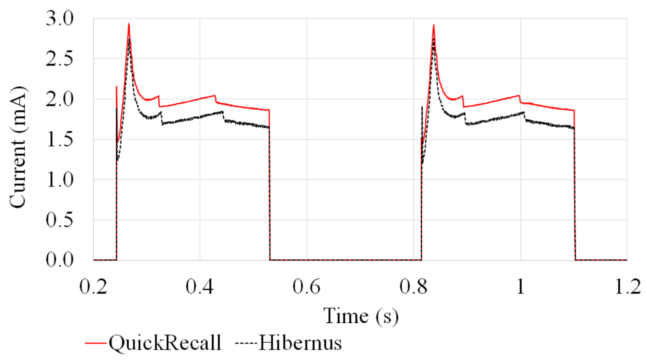

Figure 8: Current comparison

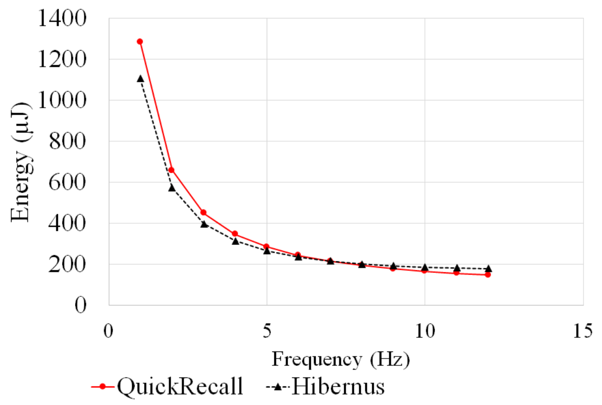

Figure 9: Energy Comparison of Hibernus and QuickRecall

encountered in real energy harvesters such as photovoltaic cell and wind turbines. As seen from the figure, the energy consumption of QuickRecall is higher than the Hibernus at lower interruption frequencies (less than 7Hz). As the frequency is increased beyond 7 Hz, the energy consumption of Hibernus increases. Thus, for the FFT application, it is energy efficient to use Hibernus for interruption frequencies lower than 7Hz, while for higher interruption frequencies, QuickRecall is more energy efficient.

5. CONCLUSIONS

In this paper, we provided a quantitative analysis of three transient computing methods. These approaches are first evaluated theoretically, and then validated with experimental measurements on the same microcontroller platform with standard FFT application. The objective is to evaluate and to compare them to identify in which conditions or scenarios one outperform the others. In particular, Mementos is useful when an application is known a priori, as it is possible to place checkpoints near critical sections (loop or function calls are just a few examples of possible strategies), enabling systems to restart execution at the beginning or after these sections. On the other hand, Hibernus and QuickRecall are completely application agnostic and they introduce a smaller time and energy overhead. However, QuickRecall can only be used with unified memory systems while Hibernus is more platform agnostic and can be used with different kind of standard systems. Apart from this, Hibernus is more energy efficient at lower interruption frequencies, while QuickRecall is more energy efficient at higher frequencies. One of the important limitations of these approaches is that they are not adaptive to the dynamics of the energy harvesting source. In future, we will investigate adaptive checkpointing approaches that takes system snapshots depending on the dynamics of the energy harvesting sources.

6. REFERENCES

[1] 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. 7(1):1–4, 2015.

[2] S. Beeby and N. White. *Energy harvesting for autonomous systems.* Artech House, 2014.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*, volume 370. Addison-wesley New York, 1987.

[4] Datasheet. MSP430FR5739, 2012. [Online] Available: http://www.ti.com/lit/ds/symlink/msp430fr5739.pdf.

[5] H. Jayakumar, A. Raha, and V. Raghunathan. QUICKRECALL: A low overhead HW/SW approach for enabling computations across power cycles in transiently powered computers. *Proc. IEEE Int. Conf. VLSI Des.*, pages 330–335, 2014.

[6] H. Kim, E. Kim, J. Choi, D. Lee, and S. Noh. Building fully functional instant on/off systems by making use of non-volatile ram. In *Consumer Electronics (ICCE), 2011 IEEE International Conference on*, pages 675–676, Jan 2011.

[7] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan. Architecture exploration for ambient energy harvesting nonvolatile processors. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 526–537, Feb 2015.

[8] P. Mitcheson, E. Yeatman, G. Rao, A. Holmes, and T. Green. Energy harvesting from human and machine motion for wireless electronic devices. *Proceedings of the IEEE*, 96(9):1457–1486, Sept 2008.

[9] S. Onkaraiah, M. Reyboz, F. Clermidy, J. Portal, M. Bocquet, C. Muller, H. Hraziia, C. Anghel, and A. Amara. Bipolar reram based non-volatile flip-flops for low-power architectures. In *New Circuits and Systems Conference (NEWCAS), 2012 IEEE 10th International*, pages 417–420, June 2012.

[10] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on RFID-scale devices. *ACM SIGPLAN Not.*, pages 159–170, 2011.

[11] G. Rebel, F. Estevez, P. Gloesekoetter, and J. M. Castillo-Secilla. Energy harvesting on human bodies. In *Smart Health*, pages 125–159. Springer, 2015.