Mementos: System Support for Long-Running Computation on RFID-Scale Devices*

Benjamin Ransford[1], Jacob Sorber[1,2], and Kevin Fu[1]

[1]Department of Computer Science, University of Massachusetts Amherst
[2]Institute for Security, Technology and Society, Dartmouth College

October 29, 2010

Abstract

Many computing systems include mechanisms designed to defend against sudden catastrophic losses of computational state, but few systems treat such losses as the common case rather than exceptional events. On the other end of the spectrum are transiently powered computing devices such as RFID tags and smart cards; these devices are typically paired with code that must complete its task under tight time constraints before running out of energy. *Mementos* is a software system that transforms general-purpose programs into interruptible computations that are protected from frequent power losses by automatic, energy-aware state checkpointing. Mementos comprises a collection of optimization passes for the LLVM compiler infrastructure and a linkable library that exercises hardware support for energy measurement while managing state checkpoints stored in nonvolatile memory. We evaluate Mementos against diverse test cases and find that, although it introduces time overhead of up to 60% in our tests versus uninstrumented code executed without power failures, it effectively spreads program execution across zero or more complete losses of power and state. Other contributions of this work include a trace-driven simulator of transiently powered RFID-scale devices.

*This tech report UM-CS-2010-060 expires March 7, 2011. The final version will appear at the *Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS 2011) that month.

1 Introduction

Recent demand for tiny, easily deployable computers has driven the development of a new class of general-purpose *transiently powered computers* that lack both batteries and wired power, operating exclusively on energy harvested from remote supplies or environmental phenomena. Examples of transiently powered computers range from *computational RFIDs* [2]—microcontroller-based devices that harvest RF delivered by readers and communicate via RFID protocols—to general-purpose batteryless sensor devices [39].

Transiently powered computing poses unique challenges to program execution. Conventional RFID tags, which are transiently powered but do not support general-purpose computation, typically use simple state machines because of limited energy available from radio-frequency (RF) harvesting. Contactless smart cards perform more complicated special-purpose computations but suffer from similar energy limitations; they offer no guarantees unless the card is placed under specific physical conditions for a certain amount of time. When energy consumption outpaces energy harvesting, these devices fail to complete their computations and must accumulate electrical charge

before restarting their computations from scratch.

Thanks to the availability of ultra-low-power microcontrollers, transiently powered devices can now perform limited computation and sensing under RFID-scale energy constraints. However, these programmable microcontrollers require so much more power than conventional RFID circuitry that RFID-scale devices are not able to fully exploit general-purpose computation. For applications such as cryptography to run on these devices [9], programs aim to finish all computation within a fixed time window before a power loss, often on the order of 100 milliseconds. A challenging problem is how to perform general-purpose computation on RFID-scale devices that lose power frequently rather than occasionally.

Mementos is a software system that combines compile-time instrumentation and run-time energy-aware state checkpointing to enable long-running computations to span power loss events. It instruments code at compile time, inserting calls to Mementos functions that estimate available energy. At run time, Mementos uses energy estimates to predict power losses and copies computational state to nonvolatile memory. It restores computational state when recovering from a power loss and prevents programs from having to restart execution from scratch.

This paper contributes the following:

1. An energy-aware state checkpointing system that splits program execution across multiple lifecycles on transiently powered RFID-scale devices. The state checkpointing system, which requires no hardware modifications to existing devices, operates automatically at run time without user intervention.

2. A suite of compile-time optimization passes that insert energy checks at control points in a program. The optimization passes implement three different instrumentation strategies for compatibility with programs of different structure.

3. A trace-driven simulator to evaluate the behavior of programs on transiently powered RFID-scale devices. The simulator, modeled after a prototype hardware device with an off-the-shelf microcontroller, takes executable code as input and simulates power loss events during runs.

The compile-time analysis and program transformation components of Mementos are built on the LLVM compiler infrastructure [18]. Our simulation of a transiently powered device is implemented as a set of enhancements to MSPsim [12] and is guided by the hardware parameters of a WISP [33] (Revision 4.1) prototype computational RFID.

Applications. Mementos is an enabling technology for long-running or computationally intensive applications on transiently powered devices. Transiently powering an RFID-scale computer is appropriate in environments that are hostile to batteries and tethered power. Applications of transiently powered computers include environmental monitoring for which batteries are difficult to replace, insect-scale wildlife tracking for which batteries are too heavy to carry, and implantable medical devices for which recharging a battery could generate heat that damages surrounding tissue. Mementos aims to enable new applications by extending the computational capabilities of transiently powered computers beyond simple programs.

An example of a long-running application that could benefit from Mementos's automatic checkpointing is compressive sensing [8]. Prototype RFID-scale devices provide sensors for physical phenomena such as temperature, acceleration, and light—all of which may exhibit informative trends over time spans larger than a few seconds. Compressive sensing maintains a set of frequently updated variables in memory that collectively represent a sparse, compressible signal, preserving the structure and information of the signal with high probability. By providing automatic state checkpointing, Mementos would enable a compressive sensing program to accumulate measurements over many lifecycles interspersed with power loss events. Section 5 includes an evaluation of Mementos on a simplified sensing application that cannot complete in a single lifecycle of our trace-driven simulator.

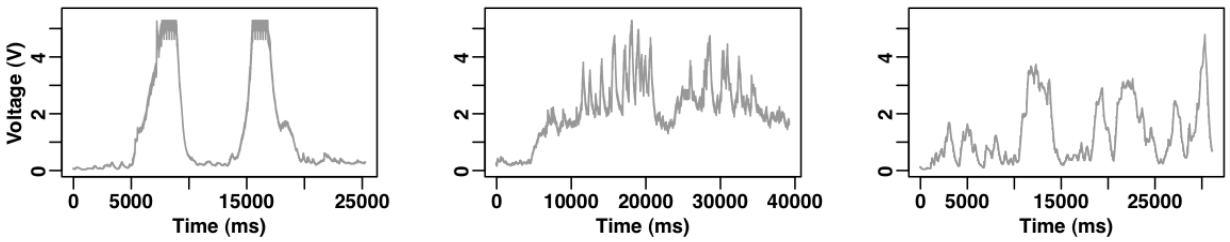

Figure 1: Because energy harvesting is fickle, energy availability may be difficult to predict on a transiently powered computer (TPC), threatening the successful completion of long-running programs. These plots show capacitor voltage on a prototype TPC during three slow human perambles within two meters of an RFID reader. When its capacitor's voltage falls below 1.8 V, this TPC loses volatile state regardless of whether it has completed its task.

2 Computing on Transient Power

Several platforms at various stages of maturity have been developed for batteryless, RFID-scale, transiently powered computing. The WISP [33], introduced in 2006, relies on an MSP430 microcontroller [37] for computation and harvests energy from (and communicates with) off-the-shelf RFID readers. The SoCWISP [27] is a WISP-like platform implemented as a custom chip with an active area of only 2.0 mm^2; it is designed to be lightweight enough for attachment to small animals and insects in flight. The BlueDevil WISP [38] is based on the WISP design and includes a similar microcontroller but uses a different analog frontend for RF harvesting and communication. EnHANTs [13] is a nascent platform that its designers intend to position (figuratively) between RFID and sensor motes. All share the goal of enabling general-purpose computation under transient power.

Transiently powered computers have been proposed for a variety of sensing and computation application, including environmental monitoring [14], activity recognition [15], and cryptographic protocols. Constantly powered mote-class devices such as the Telos mote [31] offer the same computation and sensing capabilities but import major limitations on deployability because of their size, weight, and maintenance cost—all three because of these motes' dependence on batteries. For many applications, a transiently powered device provides the benefits of programmability and general-purpose computing without the drawbacks associated with more powerful mote-class devices.

Despite their benefits, designing and deploying transiently powered systems is challenging. By definition, these systems cannot depend on a continuous supply of power. Figure 1 illustrates typical fluctuations in supply voltage that occur under RF energy harvesting. Prototype transiently powered systems at the scale of RFID tags, such as the WISP, employ capacitors that serve as short-term energy buffers. For a sense of scale, consider that a WISP's 10 μF capacitor can store roughly 100 microjoules, whereas a Telos sensor mote's two AA batteries can store over 20,000 joules—200 million times more.

The amount of energy harvested from RF, solar, and other sources varies widely and is difficult to predict [28, 40]—a problem often compounded by device mobility. The result is that, unlike traditional computing systems, transiently powered systems experience power failure and the consequent loss of computational state as a rule, not as a rare exception. Previous work using a WISP has noted that complete power failures every $\sim$100 ms are a reasonable expectation. Under these conditions, long-running programs may rarely or never be allowed to run to completion, instead restarting their work every time they regain the ability to run. In the context of transiently powered devices, we refer to such long-running programs as *Sisyphean tasks*.[1]

[1]In Greek mythology, Sisyphus was the first king of

A key to solving the problem of Sisyphean tasks on RFID-scale devices is that many general-purpose microcontrollers, notably the MSP430 found on WISP-derived devices, feature nonvolatile memory that can be written to at run time. The most common form of on-chip nonvolatile memory is *flash memory*, typically available on prototype RFID-scale devices in the amount of several kilobytes. Four complications make it nontrivial to use flash memory for checkpoint storage. First, even small flash memories are coarsely divided into segments. Each segment must be erased all at once, and erasing a segment requires energy comparable to filling the entire segment with data. Second, flash memories have a *one-way* property: once a bit is set to `0`, the only way to set it back to `1` is to erase—i.e., set to `1`—the entire segment that contains the bit in question. Another asymmetry is that flash reads are nearly as fast as volatile RAM reads, but flash writes are two orders of magnitude slower. Finally, many microcontrollers use flash memory for program storage, which limits the amount of nonvolatile storage available for other purposes.

3 Design of Mementos

The key observation motivating the design of Mementos is that it is difficult to predict the behavior of energy harvesting on a transiently powered RFID-scale computer. For example, devices that harvest energy from RFID readers are subject to fluctuations in voltage (Figure 1) that are highly dependent on the operating environment and the device's physical orientation. With the advent of *programmable*, *general-purpose* transiently powered computing comes a need for general-purpose power failure recovery mechanisms. Without general-purpose mechanisms, programs on these devices must either finish quickly—not always an option—or include potentially complicated application-specific logic to manage their own computational state. The goal of Mementos is to furnish transiently powered RFID-scale devices with system support for automatic suspension and resumption of computational state despite continual power failures. To satisfy this goal, Mementos combines *compile-time program instrumentation* with *run-time energy-aware checkpointing* to nonvolatile memory. Table 1 provides a glossary of terms we use in our discussion.

Mementos has two parts: a set of program transformation passes that insert energy-measurement code at control points in a program, and a compact library that provides state checkpointing and recovery functions. Mementos can be integrated into a project's build system via standard means (e.g., a Makefile).

Following are Mementos's high-level design goals. Given the constraints of RFID-scale devices, we consider the goals of minimizing overhead and maximizing efficiency to be self-evident. Section 5 evaluates Mementos against our design goals.

Goal #1: Split programs across multiple lifecycles. Mementos must, at run time, automatically suspend and resume programs without user intervention.

Goal #2: Take no shortcuts. Mementos must not skip portions of a computation. To remain application agnostic at run time, Mementos does not, for example, reason about quality-of-service metrics.

Goal #3: Move energy reasoning to run time. Past work has demonstrated that programmers cannot be depended upon to reason correctly about energy [34]. In the case of transiently powered RFID-scale devices, reasoning about run-time energy availability at compile time may be impossible because of inconsistent harvesting and limited computation available for prediction. Mementos implements run-time energy estimation methods, and its compile-time instrumentation inserts energy checks that prevent the programmer from having to implement complex logic to deal with changing energy conditions.

Goal #4: Require minimal support. Multiprogramming on RFID-scale devices would make Mementos's job of checkpointing state easier, but existing operating systems—including those that run wireless sensor nodes—are designed for larger-memory devices that reboot relatively infrequently; we are not aware of any transiently powered device that offers multiprogramming or even a filesystem.

Corinth and a conniving malefactor. His punishment in Tartarus was forever to repeat the task of rolling a boulder to the top of a hill only to have it roll back to the bottom.

Mementos therefore does not assume an operating system. Additionally, Mementos requires no special hardware support other than the ability to measure the voltage of the platform's energy buffer. Circuitry for voltage measurement is common on computing devices that operate on batteries or similar power supplies.

3.1 Compile-Time Instrumentation

Mementos modifies programs in two ways at compile time. First, it places *trigger points*—calls to a Mementos library function that estimates available energy—at control points in the program. Second, it wraps the program's `main()` function with code that restores execution from an available checkpoint.

The goal of Mementos's trigger-point placement is to insert enough energy measurements so that runtime energy trends are effectively sampled, but not to insert so many that measurement cost predominates over execution. If it is to be general purpose, Mementos must also be compatible with programs that are structured in different ways. To these ends, Mementos offers three different instrumentation options. In *loop-latch mode*, Mementos places a trigger point at each loop latch (the back-edge from the bottom to the top of a loop), resulting in an energy check for each iteration of each loop in the program. In *function-return mode*, Mementos places a trigger point after each call instruction, resulting in an energy check each time a function returns. In *timer-aided mode*, Mementos adds to either the loop-latch or function-return mode a hardware timer interrupt that raises a flag at predetermined intervals. Each trigger point then checks the flag and proceeds with an energy check only if the flag is up. The flag is lowered again for the next trigger point.

Besides offering three strategies for automatic trigger-point placement, Mementos exposes a simple API. A programmer can opt not to run any of Mementos's instrumentation passes and instead insert trigger points manually, simply by including a header file and placing function calls in the program. A programmer can also call Mementos's checkpointing function in a similar manner to skip energy checks entirely.

3.2 Run-time Energy Estimation

At run time, Mementos estimates the energy remaining in the device's energy buffer by measuring the energy buffer's voltage. Microcontrollers suitable for RFID-scale devices typically have on-chip analog-to-digital converters that sample voltage as a proxy for any number of environmental phenomena (e.g., temperature and physical orientation); Mementos simply makes use of this subsystem. For an ideal capacitor, the amount of energy it presently contains (E) is determined by the capacitor's present voltage (V) and its fixed capacitance (C), via the following equation: $E = CV^2/2$.

Each trigger point must quickly and accurately decide whether to initiate a state checkpoint. Since calculating energy from voltage may require computationally intensive operations such as squaring or floating-point arithmetic, Mementos uses voltage measurements directly when making checkpointing decisions: it compares the measured voltage to a *checkpoint threshold voltage*. Above this voltage, Mementos assumes that it does not need to write a state checkpoint. It interprets a voltage below the threshold as indicating that power failure is imminent and begins checkpointing state.

Ideally, program state should be saved at the last practicable opportunity before a power failure in order to minimize unsaved computation. However, unpredictable energy harvesting and the cost of saving checkpoints make perfect failure prediction infeasible. Mementos predicts future power failures *conservatively* by assuming that no energy will be added to the device's energy buffer between the trigger point and a power failure.

3.3 Run-Time Checkpointing

Mementos provides a run-time checkpointing facility in the form of a library to be linked against programs that are instrumented with trigger points. When a trigger point's voltage check initiates a checkpoint, Mementos copies relevant program state to nonvolatile memory along with some meta-information. When the device resets after a power failure, Mementos searches nonvolatile memory for a restorable

Term	Definition
Checkpoint	A copy of program state information from which execution may be restored after a reboot.
Trigger point	A check of available energy that may cause a checkpoint.
Checkpoint threshold voltage	The voltage below which Mementos turns trigger points into checkpoints.
Sisyphean task	A task that exhausts the platform's available resources each time it runs, without finishing.
Loop-latch mode	Mode in which Mementos places energy checks at loop latches.
Function-return mode	Mode in which Mementos places energy checks after call instructions.
Timer-aided mode	Mode in which Mementos performs energy checks only when a hardware timer has raised a flag.
Computational RFID	(or CRFID) A prototype example of an RFID-scale, general-purpose, transiently powered device [2].
Lifecycle	(or power lifecycle) Time during which a transiently powered device can execute code. Mementos splits computations across multiple lifecycles.

Table 1: Terms used in our discussion of Mementos.

checkpoint and, if it finds one, copies the stored state into volatile memory and resumes execution.

Several factors make checkpointing on RFID-scale devices more difficult than checkpointing on more powerful platforms. Without an operating system, Mementos must be linked into a program destined for the device. Mementos therefore shares all of the program's resources and must perform *in-place* checkpointing to capture the state of the program as it was immediately before entering the trigger point. Additionally, because of the limitations of flash memory (discussed in Section 2 and below), safely managing checkpoints is not a trivial concern.

In-place checkpointing. Most checkpointing systems in the literature are designed to run on multiprogrammed operating systems (e.g., [5]) or in hardware environments that support the issuance of commands by other devices (e.g., [26]). Mementos runs on RFID-scale devices that lack the resources to run conventional operating systems and may have no electrical connection to their environs. It interacts with its host program via function calls and shares the program's address space, stack, registers, and globals.

Flash writes are slow and energy intensive relative to volatile memory writes, so instead of blindly copying the entire contents of RAM in each checkpoint, Mementos captures only the regions of RAM that are in use at the time the trigger point is called. These comprise the stack, whose depth can be calculated via the stack pointer; the global variables, whose number and size are captured by Mementos in an analysis pass at compile time; and the register file, which is of fixed size and which includes the stack pointer and program counter. In its current form, Mementos does not capture the program's executable code because this code is typically already stored in, and executed from, nonvolatile memory.

At checkpoint time, Mementos's first action is to push all of the registers onto the stack; registers tend to change during program execution. It adjusts the stored value of the stack pointer to adjust for the function call that initiated the checkpoint, and it sets the stored value of the program counter to the return address from the checkpoint function's own stack frame. It then finds space for a new checkpoint (details below). It writes at the beginning of the free space a checkpoint size header that includes the stack depth (minus the checkpoint function's frame). It

then copies to flash the saved registers, the appropriate portion of the stack, and all globals. Finally, it writes a magic number that indicates the end of the checkpoint. The location of the magic number is trivial to calculate from the size header, which means that Mementos can detect incomplete checkpoints that are due to power failures during checkpointing.

At boot, Mementos searches for an active checkpoint (details below), then copies its contents into volatile memory. As when checkpointing, it must copy carefully so that it restores the saved state rather than a mixture of the saved state and its own state. For example, on the MSP430 architecture, it restores the register file in descending numeric order, leaving the stack pointer (`R1`) and the program counter (`R0`) for last. Restoring the program counter from the checkpoint implicitly transfers control to the program where it left off.

Checkpoint management. Unlike past systems that can exploit OS facilities to simply dump process memory to a filesystem (e.g., *libckpt* [30]), Mementos must manage its own checkpoint storage. The characteristics of flash memory require special consideration.

Mementos is designed to facilitate the execution of programs from beginning to end; as a result, once a checkpoint is successfully written to nonvolatile memory, all previous checkpoints are *superseded*. Mementos maintains at most one *active* checkpoint at any given time. At boot or when searching for free space, Mementos uses a simple active-checkpoint search algorithm: it walks a reserved region of flash memory, skipping over sequentially stored valid checkpoints (characterized by their ending with correct magic numbers) and stopping when it discovers a valid checkpoint that is followed by a byte in the erase state (`0xFF` for flash).

Flash is erasable only segment-by-segment, so to enable it to erase superseded or invalid checkpoints without destroying active checkpoints, Mementos reserves two segments of flash memory to checkpoint storage. When a checkpoint is completely written to one of these segments, it supersedes all checkpoints stored in the other, and so Mementos marks the other segment erasable by zeroing its first word—an operation that cannot be reversed in flash without erasing a whole segment. Mementos erases segments marked erasable at two times: at boot, when energy is likely to be plentiful in many scenarios, and when it fails to locate a suitable location for a new checkpoint in either segment of flash storage (i.e., during a long lifecyle when one segment is marked for erasure and the other is full of checkpoints).

4 Implementation

We use the LLVM compiler infrastructure [18] as a framework for program manipulation. LLVM allows us to formulate program manipulations as simple optimization passes that operate on LLVM's intermediate representation of a program. Hypothetically, by virtue of its operating on LLVM assembly code, Mementos can instrument programs in any language that has an LLVM compiler frontend, but we have tested it only against C programs compiled with the *clang* frontend [1].

Inspired by a prototype RFID-scale device that is transiently powered (a WISP [33], revision 4.1), we implemented Mementos for the MSP430 family of low-power microcontrollers. The WISP gathers energy and communicates on the RFID radio frequency band. Its MSP430F2132 microcontroller features 8 KB of flash memory divided into 512-byte segments; all segments of the flash memory are individually erasable but not partially erasable. The microcontroller supports clock rates of up to 16 MHz, but because lower clock rates consume less energy, the prototype's firmware keeps the clock rate below 6 MHz. The prototype's energy buffer is a 10 μF capacitor, a size chosen for its ability to charge to the microcontroller's operating voltage quickly enough to respond to an RFID reader's queries. An analog frontend comprising an antenna and harvesting circuitry feeds incoming harvested energy to the capacitor for storage. We instrumented programs that we had previously written for the WISP and compiled using LLVM.

Mementos's LLVM passes are implemented in C++ and comprise a total of 758 lines of code including whitespace, comments and header files. Me-

mentos's run-time library comprises, counting similarly, an additional 628 lines of C and inline MSP430 assembly.

Using Mementos. Integrating Mementos into an existing project targeting an MSP430 microcontroller involves a short sequence of steps and two choices. Mementos provides example Makefiles.

1. Change the compiler to LLVM (for C, the clang frontend is largely a drop-in replacement for GCC).

2. Pass the `-emit-llvm` argument to the LLVM frontend to instruct it to emit LLVM assembly instead of object code.

3. Choose a trigger point insertion strategy (Section 3) appropriate for the structure of the program being modified. For example, if the program's primary work occurs in a loop (a common paradigm, especially in sensing applications, cryptography, compression, and so on), it may be appropriate to choose Mementos's loop-latch mode. Function-return mode may be appropriate for programs that consist of many function calls. Timer-aided mode may be appropriate for event-driven programs.

4. Call LLVM's `opt` tool to run the Mementos optimization passes for the desired trigger point insertion strategy.

5. Choose a checkpoint threshold voltage (Section 5).

6. Compile Mementos's C source using a similar sequence of LLVM commands, this time omitting the extra optimization passes.

7. Call LLVM's linker, `llvm-ld`, to link the program to Mementos.

8. Use LLVM's `llc` tool to translate the linked LLVM assembly program to target-specific assembly.

9. Use an appropriate target-specific toolchain to generate an executable.

We provide a simulator based on *MSPsim* [12] that a programmer can use to debug compiled programs and evaluate Mementos's behavior. Section 5 details our use of this simulator for evaluation.

Mementos is available for download via the first author's web page.

5 Evaluation

In this section, we evaluate Mementos's ability to correctly and efficiently preserve computational state across frequent power failures. We replay measured energy conditions (10 traces) using a cycle-accurate trace-driven simulator, to observe the impact of checkpointing strategy, voltage threshold tuning, and application workload on the efficiency and overhead of Mementos. Finally, we discuss a variety of techniques—including compression and run-time adaptation—for improving Mementos's performance.

5.1 Methodology and Tools

Mementos is designed with RFID-scale devices in mind, so we developed a flexible testbed around a simulated microcontroller modeling the one found on a real device (a WISP [33]; see Section 4). The WISP's hardware parameters guide the design of our simulations.

We augmented MSPsim [12], a cycle-accurate MSP430 simulator that accepts MSP430 ELF binaries, with a simulated capacitor that obeys the basic capacitor equations for charging and discharging. The simulated capacitor stops and restarts execution whenever the capacitor voltage falls below the microcontroller's minimum operating level (1.8 V) or returns to an operable level from an energy shortfall, respectively; it is typical for embedded hardware to have protection circuitry that prevents components from operating outside their specified voltage ranges. We added to MSPsim a notion of electrical current, which governs the speed at which a capacitor's energy is depleted, and associated each of the microcontroller's operating modes with current values we measured from a hardware WISP's microcontroller using a multimeter. We made other minor changes

to MSPsim to simulate hardware failures (e.g., preserving nonvolatile memory contents across resets).

In order to simulate RFID-scale energy harvesting, we extended MSPsim to accept voltage traces recorded using real hardware. Our simulation takes (*time, voltage*) pairs and adds energy to the simulated capacitor when the voltage in the trace increases. We isolated a WISP's radio-harvesting analog frontend and attached it to a resistor that approximated the load of the WISP's microcontroller during active computation. We recorded ten voltage traces representing different patterns of motion near an RFID reader. Figure 1 shows several example voltage traces.

Test cases. Our evaluation of Mementos considers three test cases representing common tasks for low-power embedded systems.

The `sense` test case takes 64 consecutive analog-to-digital converter samples of a simulated accelerometer and computes the minimum, maximum, mean, and standard deviation of the samples, then stores these statistics to nonvolatile memory. Such computations are common in sensing applications that sample environmental phenomena. The `sense` program consists of two platform-specific functions—`setup()` and `sense()`—and several loops to perform computations (including division and square root). In loop-latch mode, Mementos instruments three loop latches, one in the sensing function and two in statistical computations. In function-return mode, Mementos instruments four function returns, one for each of the aforementioned platform-specific functions and one each for division and square root. The median and mean checkpoint size for the `sense` test case was 204 bytes—large relative to RAM size because of the size of the global array that holds sensor readings.

The `crc` test case computes a CRC16–CCITT checksum over 2 KB of onboard nonvolatile memory. Such a task is typical for devices that check the integrity of their own firmware, for example. The `crc` program consists of a loop of eight calls to a CRC function that checksums 256 bytes per call. In loop-latch mode, Mementos instruments the loop of eight calls as well as the outer and inner loops of the CRC computation. In function-return mode, Mementos instruments only the CRC function's return. The mean and median checkpoint size for the `crc` test case were 60 bytes and 60 bytes, respectively.

The `modpow` test case performs eight exponentiations of 16-bit integers modulo a 16-bit prime number. Modular exponentiation is a common task in public-key cryptography, though typically with numbers much larger than 16 bits; lacking a large-integer library, we did not attempt to fully implement a public-key algorithm such as RSA. The `modpow` program consists of a main loop that calls a modular exponentiation subroutine and a loop within modpow. Mementos instruments both of these loops in loop-latch mode; in function-return mode it instruments the function call within the main loop. The mean and median checkpoint size for the `modpow` test case were 95.7 bytes and 98 bytes, respectively.

We ran each of the three test cases against many configurations of Mementos, each differing by a small change in one configuration variable (e.g., checkpoint threshold voltage). Our testbed compiles each test case with each of Mementos's instrumentation strategies—loop-latch instrumentation, function-return instrumentation, and timer-aided loop-latch instrumentation—producing separate executables. For each such executable, the testbed sets the checkpointing threshold to a succession of values from 2.0 to 3.5 volts, then runs the resulting program against an energy trace in MSPsim. For the timer-aided loop-latch version, the testbed additionally tests a variety of timer intervals from $5,000$ to $90,000$ cycles (between 5 and 90 ms, the latter being roughly the time the simulated capacitor takes to decay from 4.5 V to 1.8 V under active computation).

5.2 Performance and Overhead

Mementos protects computations from failures caused by power loss, but it imposes some overhead in terms of time and storage space. This section details our simulation results, then characterizes and measures the overhead Mementos imposes.

5.2.1 Correctness

The key feature of Mementos is that it enables computations to complete despite intervening power losses. Figure 2 illustrates the operation of Mementos as it supervises the execution of a `crc` test case from beginning to end despite 22 such resets in our simulator; it also zooms in on a single power lifecycle (reset–compute–reset) from the same run.

To enable the CPU to properly resume execution after a reset, Mementos's checkpoints must capture all relevant state (registers, stack, and globals) as it was before checkpointing began. At run time, the simulator watches for entry to and exit from the checkpointing function. Immediately before the checkpointing function runs, the simulator captures a snapshot of the CPU's register file and RAM, a superset of what Mementos captures. When the checkpointing function returns after saving a checkpoint, the simulator compares the saved checkpoint against its pre-checkpoint snapshot. If the saved checkpoint contains incorrect information, the simulator halts execution. In a full run-through of Mementos's test suite against all test cases, we observed no such halts.

Because it may suffer power loss during a checkpointing operation, Mementos exhibits defensive behavior that ensures correctness at a cost of time—i.e., its precautions err on the conservative side and may increase the amount of redundant computation during a complete execution. Mementos's first precaution is that it writes checkpoints *head first* and *tail last*: the first word of data it writes to nonvolatile memory contains enough length information for a complete checkpoint to be reconstructed and an incomplete checkpoint to be detected; the last word it writes is the magic number that ends every valid checkpoint. Second, if Mementos detects an incomplete checkpoint during recovery or next-checkpoint location, it refuses to write any more information to the containing segment of nonvolatile memory and marks the segment for deletion. Mementos erases such marked segments immediately after boot when energy is most likely to be plentiful.

Our simulator's accuracy derives in part from MSPsim's cycle-for-cycle simulation of an MSP430 microcontroller. We manually verified that MSPsim counted a correct number of cycles for each class of MSP430 instruction, and after completing an unfinished portion of MSPsim we confirmed that flash write and erase timings were accurate. Mementos checks capacitor voltage by reading a special memory address; we confirmed that the simulated current draw and read timing for that address were the same as we measured on a real MSP430. As for the simulated capacitor, we tested it under a suite of simulated electrical currents (drawn from our measurements of a real MSP430 in its various modes) and confirmed that its decay time under each regime was accurate to within 5 ms.

In some cases, a program instrumented with Mementos performs no better than an uninstrumented program. For example, when the checkpoint threshold voltage is too close to the power loss threshold voltage (i.e., the shaded area depicted in Figure 2 becomes too narrow) and the platform does not concurrently gather enough energy to offset the cost of the checkpointing operation, Mementos's checkpointing operations always fail and the program must start from the beginning in every lifecycle. Under such conditions the program may never complete. While this behavior is undesirable, we do not consider it incorrect because the energy supply is not under Mementos's control. We discuss the effect of threshold tuning in Section 5.2.2 and suggest some appropriate adaptations of Mementos in Section 5.3.

5.2.2 Efficiency, Tuning, and Overhead

Mementos has two tunable parameters: checkpoint threshold voltage and checkpoint timer interval. Checkpoint threshold voltage (V_{thresh}) refers to the voltage below which Mementos takes checkpoints every time it encounters a trigger point. Checkpoint timer interval (T_{chk}), which applies only in timer-aided checkpointing mode, refers to the interval at which a timer interrupt (added by Mementos) raises a flag indicating that a checkpoint should be taken at the next trigger point if the voltage is below the checkpoint threshold voltage. Each parameter affects two key metrics: Mementos's share of the total number of CPU cycles required to finish the program, and the amount of waste from start to finish. For the

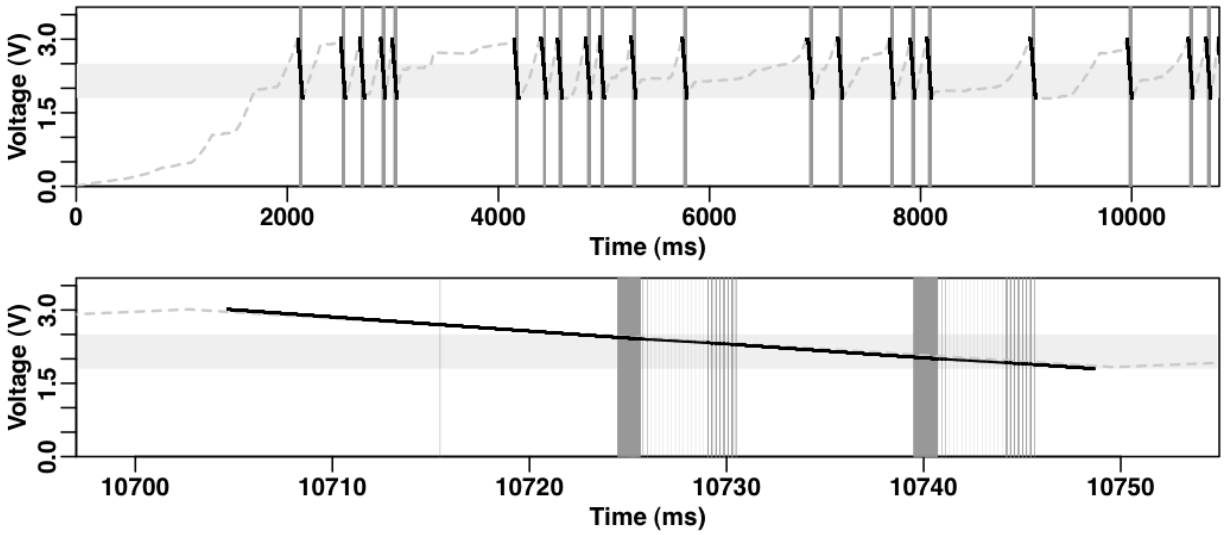

Figure 2: Simulated voltage versus time as Mementos spreads a long-running computation across 23 power lifecycles (22 resets). The simulated capacitor charges (dotted gray line) according to an input trace and discharges (solid black line) during computation and storage. When capacitor voltage falls between a voltage threshold (here 2.5 V) and the CPU's reset threshold (1.8 V—shaded region), energy checks trigger checkpointing. The bottom plot highlights a single power lifecycle from the top plot. Mementos uses the CPU (vertical lines) to check energy, find space for checkpoints, collect state, and write state to flash.

purpose of quantifying redundant computation, we define *waste* or *wasted work* as computation done *before* power failure but *after* booting or completing a checkpoint, whichever comes later. Intuitively, there is no value in work whose results are not saved.

Table 2 shows the relationships among V_{thresh}, Mementos's share of CPU cycles, and waste for a single test case (`sense`) instrumented in two different ways (loop-latch mode and function-return mode). Without Mementos instrumentation, the `sense` test case requires 122,285 CPU cycles to complete. However, when run against a voltage trace like those shown in Figure 1, the uninstrumented program cannot complete because it never receives enough energy to run for that many cycles; this uninstrumented program satisfies our definition of a *Sisyphean task*. Mementos spreads the otherwise Sisyphean task across two or more lifecycles, although it increases the total number of CPU cycles needed for program completion. In Mementos's loop-latch and function-return modes, the number of CPU cycles increases by a factor of between 2.8 and 55.2, depending on the value of V_{thresh} chosen at compile time.

Table 3 shows the effects of choosing the timer-aided checkpointing strategy in concert with loop latch instrumentation of the `sense` test case. The missing rows in Table 3 illustrate that some timer values are infelicitous with respect to checkpointing this program. Because timer interrupts cause checkpoints only at trigger points encountered when voltage is below the checkpoint threshold voltage, multiples of the timer interval may simply not fall within the appropriate region.

Practically useful values for the checkpoint threshold voltage V_{thresh} are bounded below by the power loss threshold voltage—1.8 V in the case of our simulator (and the minimum specified operating voltage of an MSP430)—and the minimum checkpoint size. Programs that keep more global state, for example, will require higher V_{thresh} values to allow checkpoints to complete. A practical upper bound is the *wakeup threshold voltage* of the platform. In our simulator, V_{thresh} values of 3.0 V and above elicit identical behavior because the simulator restarts the CPU as soon as it charges the capacitor to 3.0 V. As V_{thresh} increases, the amount of computation doable between boot and the first checkpoint decreases, so Mementos's share of computation increases.

Overhead. Although Mementos endows RFID-scale devices with the ability to split computations across power failure events, it adds run-time overhead in terms of time and code space.

Mementos's impact on execution time is smallest when energy is plentiful, such as when the microcontroller is in physical contact with a power supply. We simulated physical contact by holding the simulated capacitor's voltage above the checkpoint threshold voltage. Table 4 shows these best-case execution times for three variants of each test case. For the `sense` and `modpow` test cases, Mementos adds only a few thousand CPU cycles over uninstrumented versions' execution time. For the `crc` test case, we observe a hazard of loop latch instrumentation: when the loop body is small and executed many times, as is the case for the CRC checksum, allowing Mementos to instrument *every* loop latch results in a significant slowdown.

Mementos allows programmers to selectively disable instrumentation for sections of code. Appending the token `_mnotp` (mnemonic: "Mementos, no trigger points!") to any function's name causes Mementos to skip the function, i.e., not instrument its loops or a return from it. A programmer can disable instrumentation even for functions that are to be inlined, which means she can direct Mementos to ignore any piece of code she wishes. For the `crc` test case, a slight restructuring—turning off instrumentation for the CRC's nested for-loops by putting them inside an `_mnotp` function, and calling the CRC function more often on smaller chunks of data—reduces Mementos's best-case time overhead in loop-latch mode by over 280,000 cycles, from 51.0% to 0.7%.

Mementos adds space overhead in two ways: by increasing code size and by reserving two flash segments (1 KB on the MSP430) for checkpoint storage. Without compiler optimizations for code size, Mementos increases executable size by a constant amount (just under 2.4 KB) plus several bytes per instrumentation point (for inserted function calls).

V_{thresh}	Cycles	L	Cyc_{M} (% Cycles)	Waste (% Cycles)
Baseline (uninstrumented, unlimited energy)				
—	122,285	1	—	0 (0)
Uninstrumented, vs. voltage trace				
—	DNF	—	—	— (100)
Loop latches instrumented, vs. voltage trace				
≤ 2.3	DNF	—	—	— (100)
2.4	469,112	8	38.9	4.9
2.5	627,232	9	43.0	5.0
2.6	1,476,209	16	41.8	7.2
2.7	1,316,502	17	56.3	12.8
2.8	2,119,688	26	57.3	16.1
2.9	6,871,676	73	56.6	13.2
≥ 3.0	DNF	—	—	—
Function returns instrumented, vs. voltage trace				
≤ 2.3	DNF	—	—	— (100)
2.4	2,325,644	37	44.8	2.6
2.5	624,123	9	43.1	5.1
2.6	799,844	11	47.2	8.2
2.7	1,381,695	16	47.0	11.4
2.8	1,831,248	24	60.7	17.0
≥ 2.9	DNF	—	—	—

Table 2: Choice of checkpoint threshold voltage V_{thresh} influences the run-time behavior of two variants of the `sense` test case under Mementos. When V_{thresh} is too high, frequent checkpoints mean that Mementos dominates CPU usage and the test case makes little or no progress during each power lifecycle. When V_{thresh} is too low, the test case fails to terminate (*DNF*) because checkpointing begins too to write a complete checkpoint. In between, decreasing V_{thresh} tends to result in fewer CPU cycles and power lifecycles (L) to completion, a lesser share (Cyc_{M}) of CPU cycles used by Mementos, and decreased waste (as defined in Section 5.2.2). We took these measurements in simulation against the rightmost trace in Figure 1.

5.3 Improvements

We suggest several partially implemented or unimplemented improvements to Mementos that may improve its performance.

Techniques to reduce trigger point frequency. As we observed above for the `crc` test case, Mementos's loop latch instrumentation can result in excessively frequent trigger points when applied to loops with small bodies and large trip counts. Detecting small loop bodies and large trip counts, whether via static analysis or profiling or a combination, may prove useful toward reducing Mementos's share of CPU cycles.

Compression. Reducing checkpoint sizes has been a concern for previous checkpointing systems; past approaches have included memory exclusion [29] and straightforward file compression via external programs. On an RFID-scale device with flash memory that is expensive to write and erase, Mementos should minimize checkpoint sizes to minimize the cost of writing them (and the amortized cost of erasing them). However, Mementos is designed to run without an operating system or filesystem, and we found that most implementations of well-known compres-

T_{int}	Best				Average		
(ms)	V_t	Cycles	C_M	W	Cycles	C_M	W
10	2.6	403,980	24.7	5.8	2,504,952	45.6	11.2
20	2.8	482,832	48.5	14.8	1,216,910	42.4	9.9
40	—	—	—	—	—	—	—
60	—	—	—	—	—	—	—
80	2.6	415,481	54.0	13.1	415,481	54.0	13.1
100	—	—	—	—	—	—	—

Table 3: In timer-aided checkpointing mode, Mementos raises a flag every T_{int} cycles to indicate that a checkpoint should occur at the next opportunity if the voltage is below the checkpoint threshold voltage V_t. The table shows the best-case number of CPU cycles to completion (*Cycles*), the percentage C_M of those cycles that occurred inside Mementos during that run, and the percentage W of cycles wasted during that run. For each timer interval, we also show the average values over all voltages for which the program completed.

Test case variant	**Cycles**	**Overhead** (% cycles)
`sense` (Uninstrumented)	122,285	—
`sense+latch`	125,579	2.7
`sense+return`	123,750	1.2
`sense+timer`	126,214	3.2
`crc` (Uninstrumented)	573,925	—
`crc+latch`	866,854	51.0
`crc+return`	574,442	0.1
`crc+timer`	906,162	57.9
`modpow` (Uninstrumented)	478,175	—
`modpow+latch`	479,622	0.3
`modpow+return`	478,915	0.2
`modpow+timer`	480,440	0.5

Table 4: Run-time overhead of Mementos under plentiful energy (simulated capacitor held at a high voltage).

sion algorithms were too large to fit in our devices' limited code space. We have partially implemented several custom compression schemes.

Because many programs do not use all available registers, one promising but not fully implemented scheme compresses the register file by using a 16-bit bitmask to indicate which of the CPU's 16 registers are zero valued. During checkpointing, Mementos walks the register file, builds the bitmap, and avoids storing any registers that are zero valued.

We have also considered compressing full checkpoints instead of just the register file; all of the options predictably traded checkpoint size for run time. Our simulator saves checkpoints to files as it validates them, so we used checkpoint files as inputs to compression algorithms running in a separate MSP430 simulator. We implemented a reduced variant of the WK compression algorithm [16] but found that, while it reduced checkpoint sizes by an average of 55% for the `crc` example, it required 3.5 times as many CPU cycles as it would have taken to write the full checkpoint to flash. We implemented a variant of the popular LZ compression algorithm and found that it reduced checkpoint sizes less than WK (30%) and was 18 times slower than simply writing the checkpoint to flash.

A third type of compression is incremental compression of checkpoints. We have not yet implemented incremental compression because of the complexity of doing so.

Run-time adaptations. As the rest of Section 5 illustrates in detail, compile-time tuning of Mementos's parameters can significantly change its behavior. We have designed but not implemented schemes by which Mementos could adapt its behavior at run time based on its measurement of key metrics. For example, to avoid executing time- and energy-intensive flash erasures at the beginning of lifecycles, Mementos could decrease the frequency of failed checkpoints by including in each checkpoint header the current value of the checkpoint threshold voltage. If Memen-

tos were to notice an aborted checkpoint, it could adjust the checkpoint threshold voltage as appropriate. A similar technique might enable Mementos to gradually minimize the amount of wasted work.

We have not designed an energy prediction model for Mementos's run-time system because we assume that such a scheme would be prohibitively time intensive. Relaxing some of our assumptions about unpredictability might lead us to develop lightweight prediction schemes—integer versions of first- and second-order voltage trend approximations, for example—that could allow Mementos to avoid checkpointing if it believes power failure is not imminent.

Interrupts instead of polling. Mementos is designed to work on prototype RFID-scale devices without any modification to their hardware, so for maximum flexibility it polls for supply voltage. However, Mementos's awareness of voltage could be improved by adding circuits that fire interrupts at salient voltage levels. Mementos's checkpointing and restoration functions work the same way regardless of how Mementos detects available energy, so we expect Mementos to be easily portable to hardware with these interrupt circuits.

Sleeping when appropriate. Most microcontrollers have RAM-retention modes that retain processor state and the contents of volatile memory. Such modes typically require two orders of magnitude less electrical current than active-mode computation, which slows—but does not stop—capacitor drain. We designed Mementos to be useful when energy delivery is arbitrarily sporadic. Some energy-harvesting mechanisms, such as solar panels, exhibit sudden or prolonged periods of harvesting nothing; in this case, Mementos's strategy of checkpointing to nonvolatile memory would be more suitable than simply entering RAM-retention mode. However, we suspect that a hybrid approach incorporating both RAM retention and nonvolatile checkpoints would be a fruitful avenue for improvements to Mementos.

Wear leveling for flash. Mementos is designed to work with nonvolatile memory of any type, although its present implementation is specialized for characteristics of flash memory. A factor that complicates Mementos's use of flash memory is that segment erasure, an operation that Mementos uses in checkpoint maintenance, causes irreversible wear to flash cells. It is well known that flash cells can tolerate only 10,000 to 1 million erasures before becoming unusable. To mitigate the effect of its bundle management scheme on flash lifetime, Mementos could be extended to use information coding schemes to allow rewrites without erasures [6]. Future RFID-scale devices might provide nonvolatile memory in the form of phase-change memory (PCM), magneto-resistive RAM (MRAM) or ferroelectic RAM (FeRAM), all of which import fewer complications and are more forgiving with respect to erasure, but flash remains the most widespread form of nonvolatile memory in use today.

6 Discussion and Future Work

In this section, we discuss the use of Mementos in the context of several alternatives an application developer might consider. We then suggest some future extensions to Mementos.

6.1 Alternative approaches

Mementos provides automatic state checkpointing under the assumption that manual checkpointing—that is, incorporating state-saving code into applications at key junctures—is not, in the general case, appropriate for RFID-scale devices that are transiently powered. In particular, even under controlled conditions (as in Figure 1), predicting the availability of energy supplied via RF harvesting is notoriously difficult [28, 40]. Radio-based devices that behave one way in the lab tend to behave differently once deployed. Mementos allows programmers to keep program logic simple and unencumbered by manual checkpointing code. Additionally, programmers are free to manually insert calls to Mementos's checkpointing function if they want to guarantee that it runs at certain points; these calls are resolved at link time.

Another alternative is to reduce the need for checkpointing by abbreviating computations. Many existing RFID-scale devices that perform nontrivial com-

putations, such as contactless smart cards, require a minimum exposure time to ensure that their computations have enough time to finish. Mementos relaxes this requirement by splitting program execution across power failures. If an application requires more time or energy than is typically available in one lifecycle, it is appropriate to use Mementos. Instead of suggesting that Mementos should be back-ported onto existing devices that already perform brief computations, we propose that Mementos enables new classes of applications that perform onboard computation at RFID scale without batteries.

Recent work [3, 25] has studied the effect of program modifications that shorten computations, often for the purpose of saving energy, but may be precise only within a quality-of-service (QoS) bound. These approaches are especially appropriate when the application's computations are already lossy or noisy. Mementos cannot in general assume that omitting part of the computation is acceptable, so it executes every program instruction at least once; this may make Mementos unappealing if the application must merely meet a QoS constraint.

One might ask why computation should be performed on transiently powered devices at all. Our answer is twofold: first, it is true that, for RFID-scale devices powered by radio signals, the device transmitting the signals is likely to have access to more computing resources than the devices it powers. (RFID readers, for example, often connect to PCs or run modified Linux kernels themselves.) However, outsourcing computation has significant security and privacy implications; even if a cryptographically secure mechanism existed, the computational cost of its cryptographic operations could overwhelm a constrained RFID-scale device. A second reason is that, to save power, prototype RFID-scale devices use low-throughput radio hardware or facilitate low-throughput radio communication in software. The difficulty of outsourcing computation via a low-throughput channel leads us to believe that there will continue to be a need for computation that occurs on board transiently powered devices.

Alternative hardware designs. Mementos's simulator models an energy-harvesting computer with parameters that match a specific prototype RFID-scale device–viz. a computational RFID [2]—but these parameters are trivially adjustable in software. Gummeson et al. [14] discuss the scaling effects of changing various hardware parameters, such as capacitor size and harvesting technique, on a related RFID-scale device.

Other natural scaling effects are worth considering. Trends in the density of traditional batteries suggest that there is no analogue to Moore's Law for batteries [28]. Alternative types, such as thin-film batteries, present an appealing alternative for RFID-scale devices [13] but are not yet widely available. A fundamental limitation of large reserves of energy is that they require time to fill; it is therefore reasonable to believe that responsive RFID-scale devices will continue to have small energy buffers.

6.2 Future Work

An obvious extension to the present work is an evalution of Mementos on real hardware instead of in simulation. While we have measured our simulator's individual components and found them to represent reality faithfully, we lack end-to-end measurements from real-world applications.

We also plan to implement most of the improvements listed in Section 5.3, particularly checkpoint compression, run-time adaptation of Mementos's behavior via introspection, and an extension of Mementos to support the strategic use of the MSP430's low-power modes.

An ancillary contribution of the Mementos project has been extensive testing of and bug fixes to LLVM's MSP430 backend. Ongoing work in this vein will involve further testing and enhancement of appropriate intrinsic functions.

7 Related Work

There has been a wealth of research on checkpointing at various levels of computer systems. Most of the related work on program checkpointing adopts a similar (if broader) approach to Mementos's: capture relevant program state. A key difference between Mementos and previous work is that, because of the

kinds of devices for which it is designed, Mementos must consider catastrophic failure to be the *common case* rather than an *occasional event*. We group related work into general checkpointing papers and papers related to tolerating failures on small-scale devices.

Checkpointing. We borrow our definition of *checkpointing* from Bernstein et al. [4], who define checkpointing as "an activity that writes information to stable storage during normal operation in order to reduce the amount of work [the system] has to do after a failure." Work on automatic checkpointing has long focused on providing insurance against occasional failures. Systems in the 1980s and 1990s explored checkpointing for distributed systems [24, 23, 17], particularly for process migration or high-assurance computing. Checkpointing is especially useful for systems that handle precious data or make promises about fidelity, such as databases [4, 22] or filesystems [32, 36].

Plank et al. [30] discuss checkpointing strategies in detail. Their portable *libckpt* library for UNIX implements both automatic (periodic, checkpoint-on-write) and user-directed checkpointing strategies. In the terminology of *libckpt*, Mementos implements *sequential* checkpointing, wherein the checkpointing procedure stops execution of the main program to capture its state. Like Mementos in timer-aided mode, *libckpt* automatically captures application state (registers and RAM) at a predefined frequency. Unlike Mementos, *libckpt* also supports *incremental* checkpointing by using page protection mechanisms to keep track of pages dirtied since the last checkpoint operation. We have not implemented a similar system because Mementos is designed to run directly on hardware.

Previous work has considered the use of static analysis and compile-time modifications to facilitate checkpointing. Compiler-assisted checkpointing systems [20, 21] require users to insert checkpointing cues into programs, unlike Mementos, although Mementos shares the notion of using compile-time instrumentation to make programs amenable to checkpointing. The Porch source-to-source compiler [35] enables programs to be suspended, migrated and resumed on different architectures. Porch uses compile-time analysis to generate program-specific checkpoint and resume functions specific to each possible stopping point. We consider Porch to be too heavyweight for Mementos's target platforms (owing to its lofty goals) although the checkpointing mechanism is similar.

Also relevant, perhaps surprisingly, are checkpointing systems that work on large-scale computers. These computers must tolerate frequent node failures, so job migration is a key feature. Bronevetsky et al. [5] propose a compile- and run-time system that modifies shared-memory programs and coordinates checkpointing and recovery among application threads. Their compiler techniques are essentially the same as Porch's and import the same differences versus Mementos.

Checkpointing for small-scale devices. Recent work in sensor networks considers the problem of whole-network checkpointing. In fact, Österlind et al. have used MSPsim as part of a whole-network checkpointing system [26] that facilitates experimentation on sensor networks with continuously powered sensor nodes running the Contiki operating system [11]. They have implemented a checkpointing mechanism that saves the entire contents of a sensor node's memory, plus the state of several peripherals, via the node's serial port. A master node freezes and restores nodes by issuing serial-port commands to them. This checkpointing mechanism, essentially a memory dump performed by an OS thread, is considerably simpler than Mementos, but it is not applicable to the same kinds of devices. Mementos is designed to make local decisions about when and what to checkpoint, and its goal is to enable computing despite frequent power failures rather than migration between testbeds.

The Neutron operating system for sensor network nodes [10], based on TinyOS [19], performs a function similar to that of Mementos. Neutron uses compile-time analysis and run-time program supervision to isolate and restart misbehaving components, including the TinyOS kernel. It relies on TinyOS's multithreading to isolate groups of components (called *recovery units*) from one another. Neutron allows programmers to mark "precious" state that must be preserved across restarts of recovery units—but not

across hardware reboots. Neutron is able to infer recovery unit boundaries at compile time via a simple examination of a TinyOS application's component graph. Mementos assumes no operating system, operates on LLVM intermediate code rather than TinyOS programs, does not require users to mark precious state, and must work despite reboots.

Specific to RFID-scale devices, Buettner et al. [7] describe WISP-based *RFID sensor networks* (RSNs) and the difficulty of predicting energy availability. They suggest, but do not implement, program splitting as an approach to the execution of large programs.

8 Conclusions

Transiently powered RFID-scale devices enable general-purpose computation in scenarios where energy is scarce. However, the lack of a steady supply of energy results in frequent complete losses of power and state. Today, programmers either write short programs or hand-tune assembly code to ensure that computation finishes before a power loss—severely limiting the application space for these devices and making programming cumbersome and error prone.

Mementos addresses the challenge of enabling long-running programs to make steady progress on transiently powered devices. It instruments programs with energy checks at compile time and provides automatic state checkpointing and recovery at run time.

Acknowledgments

This material is supported by a Sloan Research Fellowship and the NSF under CNS-0627529, CNS-0845874, NSF CNS-0923313, and a Graduate Research Fellowship. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

We thank John Brattin for code and discussions; Shane Clark for help with simulation; Mark Corner for providing resources; Chris Erway and Quinn Stewart for feedback on drafts; Edwin Foo, Anton Korobeynikov, Dmitriy Matveev, and John Regehr for help with LLVM's MSP430 backend; Scott Kaplan for discussions on compression; Joshua Smith and Alanson Sample at Intel Labs Seattle for providing the WISP over the last three years; and John Tuttle for conducting measurements.

References

[1] clang: a C language family frontend for LLVM. `http://clang.llvm.org/`.

[2] *Proceedings of the First Workshop on Wirelessly Powered Sensor Networks and Computational RFID*, Berkeley, CA, November 2009.

[3] W. Baek and T. Chilimbi. Green: A system for supporting energy-conscious programming using principled abstractions. Technical Report MSR-TR-2009-89, Microsoft Research, July 2009.

[4] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[5] G. Bronevetsky, D. Marques, K. Pingali, P. K. Szwed, and M. Schulz. Application-level checkpointing for shared memory programs. In S. Mukherjee and K. S. McKinley, editors, *ASPLOS*, pages 235–247. ACM, 2004.

[6] J. Bruck, A. Vardy, A. Jiang, E. Yaakobi, J. Wolf, R. Mateescu, and P. Siegel. Storage coding for wear leveling in flash memories. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1229 –1233, June 2009.

[7] M. Buettner, B. Greenstein, A. Sample, J. R. Smith, and D. Wetherall. Revisiting smart dust with RFID sensor networks. In *Proceedings of the 7th ACM Workshop on Hot Topics in Networks (HotNets-VII)*, October 2008.

[8] E. Candès, J. Romberg, and T. Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency informa-

tion. *IEEE Transactions on Information Theory*, 52(2):489–509, February 2006.

[9] H.-J. Chae, D. J. Yeager, J. R. Smith, and K. Fu. Maximalist cryptography and computation on the WISP UHF RFID tag. In *Proceedings of the Conference on RFID Security*, July 2007.

[10] Y. Chen, O. Gnawali, M. Kazandjieva, P. Levis, and J. Regehr. Surviving sensor network software faults. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 235–246, New York, NY, USA, 2009. ACM.

[11] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN*, pages 455–462. IEEE Computer Society, 2004.

[12] J. Eriksson, A. Dunkels, N. Finne, F. Österlind, and T. Voigt. Mspsim – an extensible simulator for msp430-equipped sensor boards. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, Jan. 2007.

[13] M. Gorlatova, P. Kinget, I. Kymissis, D. Rubenstein, X. Wang, and G. Zussman. Challenge: ultra-low-power energy-harvesting active networked tags (EnHANTs). In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking (MobiCom '09)*, pages 253–260, 2009.

[14] J. Gummeson, S. S. Clark, K. Fu, and D. Ganesan. On the limits of effective micro-energy harvesting on mobile CRFID sensors. In *Proceedings of 8th Annual ACM/USENIX International Conference on Mobile Systems, Applications, and Services (MobiSys 2010)*, June 2010. To appear.

[15] S. Helal, H. Gellersen, and S. Consolvo, editors. *UbiComp 2009: Ubiquitous Computing, 11th International Conference, UbiComp 2009, Orlando, Florida, USA, September 30 - October 3, 2009, Proceedings*, ACM International Conference Proceeding Series. ACM, 2009.

[16] S. F. Kaplan. *Compressed Caching and Modern Virtual Memory Simulation*. PhD thesis, University of Texas at Austin, December 1999.

[17] R. Koo and S. Toueg. Checkpointing and rollback-recovery for distributed systems. In *ACM '86: Proceedings of 1986 ACM Fall joint computer conference*, pages 1150–1158, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[18] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[19] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. TinyOS: An Operating System for Sensor Networks. In *Ambient Intelligence*. Springer Verlag, 2004.

[20] C. Li and W. Fuchs. CATCH—Compiler-assisted techniques for checkpointing. In *Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, pages 74–81, 1990.

[21] C. Li, E. Stewart, and W. Fuchs. Compiler-assisted full checkpointing. *Software-practice and Experience*, 24(10):871–886, 1994.

[22] J.-L. Lin and M. H. Dunham. A survey of distributed database checkpointing. *Distributed and Parallel Databases*, 5(3):289–319, 07 1997.

[23] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor – a hunter of idle workstations. In *ICDCS*, pages 104–111, 1988.

[24] J. A. McDermid. Checkpointing and error recovery in distributed systems. In *ICDCS*, pages 271–282. IEEE Computer Society, 1981.

[25] S. Misailovic, S. Sidiroglou, H. Hoffman, and M. C. Rinard. Quality of service profiling.

In *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*, Cape Town, South Africa, May 2010.

[26] F. Österlind, A. Dunkels, T. Voigt, N. Tsiftes, J. Eriksson, and N. Finne. Sensornet checkpointing: Enabling repeatability in testbeds and realism in simulators. In *Proceedings of the 6th European Conference on Wireless Sensor Networks, EWSN 2009*, Cork, Ireland, Feb. 2009.

[27] B. Otis and D. Yeager. SoCWISP: Ultra-low Power Wireless Sensing RFID Chip. WISP Summit Workshop, 2009. Presentation.

[28] J. A. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 4(1):18–27, 2005.

[29] J. S. Plank, M. Beck, and G. Kingsley. Compiler-assisted memory exclusion for fast checkpointing. *IEEE Technical Committee on Operating Systems and Application Environments*, 7(4):10–14, Winter 1995.

[30] J. S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent checkpointing under Unix. In *Usenix Winter Technical Conference*, pages 213–223, January 1995.

[31] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, April 2005.

[32] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst.*, 10(1):26–52, 1992.

[33] A. P. Sample, D. J. Yeager, P. S. Powledge, A. V. Mamishev, and J. R. Smith. Design of an RFID-based battery-free programmable sensing platform. In *IEEE Transactions on Instrumentation and Measurement*, 2008.

[34] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. Eon: A Language and Runtime System for Perpetual Systems. In *Proceedings of The Fifth International ACM Conference on Embedded Networked Sensor Systems (SenSys)*, pages 161–174, Sydney, Australia, November 2007.

[35] V. Strumpen. Portable and fault-tolerant software systems. *IEEE Micro*, 18(5):22–32, 1998.

[36] Swaminathan Sundararaman, Sriram Subramanian, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Michael M. Swift. Membrane: Operating System Support for Restartable File Systems. In *Proceedings of the 8th Conference on File and Storage Technologies (FAST '10)*, San Jose, California, February 2010.

[37] Texas Instruments Incorporated. MSP430 Ultra-Low Power Microcontrollers. http://www.ti.com/msp430.

[38] S. Thomas, J. Teizer, and M. Reynolds. Electromagnetic energy harvesting for sensing, communication, and actuation. In *IAARC International Symposium on Automation and Robotics in Construction*, June 2010. to appear.

[39] Y. Yang, L. Wang, D. K. Noh, H. K. Le, and T. F. Abdelzaher. Solarstore: enhancing data reliability in solar-powered storage-centric sensor networks. In K. Zielinski, A. Wolisz, J. Flinn, and A. LaMarca, editors, *MobiSys*, pages 333–346. ACM, 2009.

[40] E. Yeatman. Advances in power sources for wireless sensor nodes. In *Proc. Int'l Workshop on Wearable and Implantable Body Sensor Networks (BSN '04)*, pages 20–21, April 2004.