The Hidden Challenges of Intermittent Execution

Brandon Lucia | Microsoft Research

work done in cooperation with

Ben Ransford | University of Washington
Emerging Devices Everywhere

Wearables, sensors, “Internet of Things”, medical implants, environmental monitoring, security, computational art, music, interactive clothing, smart furniture, smart carpets
Tiny Computers Everywhere!
Systems expect **reliable power**
Tethered to a power source
Energy harvesting untethers devices

- energy receiver (antenna)
- energy buffer (capacitor)
- computer (microcontroller)
- energy source
- empty space
...but energy is **intermittent**!
The Intermittent Execution Model
Thinking About Intermittently-powered Devices

Intermittence & Data Consistency
Hidden Problems Caused by Intermittence

Designing for Intermittence
Strategies for Coping with Intermittence
Running on intermittent energy

RF Available!
Available Energy

Computing (discharging only)

Time

Running on intermittent energy
Running on intermittent energy
Running on intermittent energy

Available Energy

RF Available!
Running on intermittent energy
Running on intermittent energy

Available Energy

Time

And so on

RF Available!
Available Energy

Time

“Death Line”

Computer shuts down

Computer reboots
The Intermittent Execution Model

Compute
Reboot
Compute

Time
The Intermittent Execution Model

Goal: Run programs that take longer than one green box
void main(void) {
    for (i = 1 .. 10)
        append()
}

void append()
{
    sz++
    buf[sz] = ‘a’
}
void main(void) {
    for (i = 1 .. 10) {
        append()
    }
}

void append() {
    sz++
    buf[sz] = 'a'
}
void main(void) {
    for (i = 1 .. 10) {
        append()
    }
}

void append() {
    sz++
    buf[sz] = 'a'
}

for (i = 1)
append()
for (i = 1)
append()
sz++
buf[sz] = 'a'
for (i = 2)
append()
sz++
buf[sz] = 'a'
REBOOT

REBOOT
```c
void main(void){
    for( i = 1 .. 10)
        append()
    
    void append(){
        sz++
        buf[sz] = 'a'
    }
}
```
We can model intermittence as a control-flow problem
void main(void){
    for( i = 1 .. 10)
        append()
}

void append(){
    sz++
    buf[sz] = 'a'
}
Control-flow Graph

void main(void){
    for( i = 1 .. 10)
        append()
}

void append(){
    sz++
    buf[sz] = 'a'
}

Failure Induces **Non-Local, Implicit** Control-flow
Intermittent Execution

Challenge #1:

Implicit, non-local control-flow “back in time” on reboot.

Failure Induces

Non-Local, Implicit Control-flow
Mixture of Volatile & Non-volatile State

volatile memory (e.g., DRAM, SRAM registers)

Access latencies growing similar w/ new technology

non-volatile memory (e.g., Flash, FRAM)
Reboots **clear** volatile state and **preserve** non-volatile state

Volatile State

Non-volatile State

Time

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

Assume non-volatile storage on device
[Flash & FRAM on TI MSP430]
Periodically store checkpoint of regs, stack, globals.

Possible Checkpoint!

for( i = …)
call append()
<loop>
buf[sz] = ‘a’
SZ++
end of main()
for( i = …)
    call append()
<loop>
buf[sz] = ‘a’
SZ++
end of main()
The Big Idea

The way we think about programs does not match the intermittent execution model.
The Big Idea

The way we think about programs does not match the intermittent execution model especially when we combine non-volatile and volatile memory!
The Intermittent Execution Model
Thinking About Intermittently-powered Devices

Intermittence & Data Consistency
Hidden Problems Caused by Intermittence

Designing for Intermittence
Strategies for Coping with Intermittence
The Intermittent Execution Model
Thinking About Intermittently-powered Devices

Intermittence & Data Consistency
Hidden Problems Caused by Intermittence

Designing for Intermittence
Strategies for Coping with Intermittence
Why does **Intermittent Execution Challenge #1** matter?

Non-volatile storage is continuously accessible in user code.

Can put `sz` & `buf` in Non-Volatile storage

- registers, stack vars, global vars
- Non-volatile storage records checkpoints periodically!

Non-volatile storage is continuously accessible in user code.

**IEC#1** + Non-Volatile Storage = Data Consistency Violations!
Intermittent Execution Challenge #2: Hidden Atomicity Violations

```c
main()
append()
for (i = ...)
call append()
<loop>
buf[bufsz] = 'a'
sz++
end of main()
buf[sz] = -1
```

Diagram:
- `buf` and `sz`
- `sz++`
- `buf[bufsz] = 'a'`
Intermittent Execution Challenge #2: Hidden Atomicity Violations

```c
main()
append()
for( i = ...)
call append()
buf[sz] = 'a'
end of main()
```

```
for( i = 1)
call append()
sz++
buf[sz] = 'a'
```

```
Intermittent Execution Challenge #2: Hidden Atomicity Violations
```
Intermittent Execution Challenge #2: Hidden Atomicity Violations

Error: ‘a’ is appended to the wrong entry in buf
Intermittent Execution Challenge #2: Hidden Atomicity Violations

Atomicity Violation leaves \( sz \) and \( buf \) mutually inconsistent

Note: Restarting from checkpoint.
Intermittent Execution Challenge #3: Hidden Loops ("Idempotence Violations")

```plaintext
main()
append()
for( i = …)
call append()
<loop>
dsz++
buf[sz] = 'a'
buf[sz] = -1
end of main()
```
Intermittent Execution Challenge #3: Hidden Loops
Intermittent Execution Challenge #3: Hidden Loops (“Idempotence Violations”)

Error: ’a’ is appended to buf twice on the i = 1 iteration
Intermittent Execution Challenge #3: Hidden Loops ("Idempotence Violations")

Stuck in an implicit loop for $i = 1$
Intermittent Execution Challenge #3: Hidden Loops (“Idempotence Violations”)

The data are inconsistent with the checkpoints execution context.
Intermittent Execution Challenge #1: Implicit, non-local control flow

Intermittent Execution Challenge #2: Hidden Atomicity Violations

Intermittent Execution Challenge #3: Hidden Loops ("Idempotence Violations")
The Intermittent Execution Model
Thinking About Intermittently-powered Devices

Intermittence & Data Consistency
Hidden Problems Caused by Intermittence

Designing for Intermittence
Strategies for Coping with Intermittence
void main(void){
    for( i = 1 .. 10)
      task_boundary()
        append()
    }

void append(){
    sz++
    buf[sz] = ‘a’
}

Statically defined task boundaries.
Dynamically defined tasks
Task boundaries are checkpoints

Key Advantage: Target of failure-induced flows now explicit in code!

Task-based Programming Makes Implicit Flow Explicit
Key Advantage: task-atomicity w.r.t. volatile and non-volatile data
Eliminates Hidden Atomicity Violations!

cf. Atomic, multi-word NV update support
Mnemosyne [ASPLOS’11]
NV-Heaps [ASPLOS’11]
for( i = 1)
call append()
buf[sz] = 'a'
sz++

for( i = 2)
call append()
sz++
buf[sz] = 'a'

Correctly Updated Only Once!

Checkpoint

Hidden Loop!

Eliminates Hidden Loops, Too!
int NV_Array[1000000];

```
for(i = 0; i < 1000000; i++)
{
    NV_Array[i]++;
}
```

task_boundary

What non-volatile data do we need to version here?

Selective Versioning with Data-flow Analysis
int NV_Array[1000000];

**task_boundary**

for (i = 0; i < 1000000; i++)
{
    NV_Array[i]++;
}

**task_boundary**

**Insight:** Must version data! Task reads & writes same NV data
Prototype Implementation

```
int NV_Array[1000000];
task boundary
for( i = 0; i < 1000000; i++){
    NV_Array[i] = i;
} task boundary
```
Energy-harvesting Platform Prototypes

WISP5 from Univ. of Washington

Custom prototype board

Applications

Activity Recognition (accelerometer+ML)

Cold-chain Equipment Monitor

Multi-granular Sensor Log

Key Result: Applications suffer errors & failures without our system support for tasks
Activity Recognition

e.g., fall detection, tremor detection
Activity Recognition
e.g., fall detection, tremor detection

Percent Error

No Support  With Tasks  No Support  With Tasks  No Support  With Tasks  No Support  With Tasks

0 10cm 20cm 40cm 60cm

15th Annual Microsoft Research Faculty Summit 2014
Activity Recognition
e.g., fall detection, tremor detection

Tasks Eliminate Error Due to Intermittence!

Percent Error

No Support  With Tasks
10cm        20cm        40cm        60cm

0  2  4  6
The Intermittent Execution Model
Thinking About Intermittently-powered Devices

Intermittence & Data Consistency
Hidden Problems Caused by Intermittence

Designing for Intermittence
Strategies for Coping with Intermittence