Reliable Intermittent Computers
Brandon Lucia | Assistant Professor, CMU ECE
ABSTRACT Research Group - https://intermittent.systems
with PhD Students: Kiwan Maeng, Emily Ruppel, Vignesh Balaji
Graham Gobieski, Milijana Surbatovich, Brad Denby, Harsh Desai
PhD Alumni: Alexei Colin
Academic Collaborators: Nathan Beckman (CMU), Michael Bond (OSU), Joseph Devietti (UPenn),...
Industry Collaborators: Zac Manchester (Stanford/KickSat)
Alanson Sample (Disney Research Pittsburgh)
Emerging Devices everywhere:

- Wearables
- Sensors
- "Internet of Things"
- Tiny satellites
- Medical implants
- Environment monitoring
- Security
- Computational art
- Interactive clothing
- Smart furniture
- Smart carpets
- Smart toilet paper
Tethered to a power source
Energy harvesting untethers devices

- energy receiver
- energy buffer (capacitor)
- computer (microcontroller)
- energy source
- empty space
No battery lifetime limitation
No environmental battery waste
No battery replacements
No bulky battery
...but energy is intermittent — and as a result — software becomes inherently unreliable.
The Intermittent Execution Model
Reasoning About Intermittently-powered Devices

Designing for Intermittence
System Support for Reliable Intermittence

Intermittent & Physically Constrained Computing Platforms
Hardware, Programming Models, System Software
Running on intermittent energy
Running on intermittent energy
Running on intermittent energy
Running on intermittent energy
Running on intermittent energy
Intermittent Operation on Harvested Energy

Available Energy

Time

Computer shuts down

Operate for e.g., 500ms

Computer reboots

Recharge for e.g., 1.2s

“Life Line”

“Death Line”
The Intermittent Execution Model
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'
}
```c
void main(void){
    for( i = 1 .. 10)
        append()
}

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

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

void append()
{
    sz++
    buf[sz] = 'a'
}
```
We can model intermittence as a control-flow problem.

Control-flow Graph

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

void append(){
    sz++
    buf[sz] = 'a'
}
Intermittent Execution Challenge:
Implicit control-flow “back in time” on reboot.

Control-flow Graph

Failure Induces Implicit Control-flow

Back in time!
Mixture of Volatile & Non-volatile State

- Volatile memory (e.g., DRAM, SRAM registers)
- Non-volatile memory (e.g., Flash, FRAM)

Access latencies growing similar w/ new technology
Reboots clear volatile state and preserve non-volatile state.
Control-flow Graph

Checkpoint introduce more complex implicit control-flow

Intermittent Execution Challenge:
Implicit control-flow “back in time” on reboot.
Interruption is a Challenge

Software that is **correct** with continuous power can be **incorrect** in an intermittent execution.
Intermittence Bug: Out-of-thin-air Values

[MSPC '14; PLDI '15]

```c
main()
append()
for( i = ...)
call append()
<loop>
buf[sz] = 'a'
buf[sz]++
ckpt
end of main()
```
Intermittence Bug: Out-of-thin-air Values

[MSPC ’14; PLDI ’15]

```
main()
append()
for( i = …)
call append()
end of main()

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

Intermittence Bug: Out-of-thin-air Values

[MSPC ’14; PLDI ’15]
Intermittence Bug: Out-of-thin-air Values [MSPC ’14; PLDI ’15]

Error: ‘a’ is appended to buf twice on the i = 1 iteration

for(i = 1)
call append()
sz++
buf[sz] = ‘a’

for(i = 1)
call append()
sz++
buf[sz] = ‘a’

buf[1] = ‘a’

sz = 1

Intermittence Bug: Out-of-thin-air Values
Intermittence Bug: Out-of-thin-air Values

[MSPC '14; PLDI '15]

Stuck in an implicit loop for i = 1
Intermittence Bug: Out-of-thin-air Values

"Out of thin air" value not permitted by any continuous execution!

Stuck in an implicit loop for \( i = 1 \)

Memory contents depend on the availability of harvestable energy and capacitor size!
Interruption Challenge: Livlock & Non-termination

[CASES ’15, ASPLOS ’16, ASPLOS ’18, CC ’18, OSDI ’18, PLDI ’19]

Energy cost: atomic span exceeds energy capacity

Code reboots indefinitely on intermittent power.

Variable E cost? Unpredictable livelock

If energy cost exceeds full capacitor charge:
never completes 1 iteration

main()

for( i = …)
call senseProcTx()

<loop>
call sense()

call process()

call transmit()

end of main()

NV

Affected code need not even modify NV storage!
**Intermittence Challenge: I/O Atomicity & Timeliness**

*[ASPLOS '18, PLDI '19]*

**Checkpoint between I/O operations that should execute together leaves I/O inconsistent**

No atomicity control w/ automatic checkpoints

Inconsistent data read at different moments in time
Intermittence Challenge: I/O Non-Idempotence & Event-Driven I/O

Re-execution of I/O operations leads to non-idempotent re-execution of part of a program

Not all I/O dependent data checkpointed

Variation in input values makes normally-idempotent writes non-idempotent

If \( X > \text{max} \) { \( \text{BLINK} = 1 \) }
Intermittent software is unreliable.

Need easy programming and reliable execution.
Reliable Intermittent Execution
Task-based Programming Models
Task-based Intermittent Execution

Chinchilla
[OSDI ’18]

Automatic Intermittent Execution without Non-termination
Task-based Intermittent Execution

Tasks have *atomic* all-or-nothing semantics.

Task boundaries *selectively* checkpoint *volatile* and *non-volatile* state.

No *implicit* control-flow!
Programmer explicitly defines tasks and task-flow graph in code

Channels *statically multiversion* data separating inputs from outputs

Tasks exchange data via abstract *channels* of non-volatile memory

Key limitation: Programmer must learn new programming model

Chain: Static data multi-versioning

[Colin, et al OOPSLA ’16]
Compiler *automatically privatizes* data creating a copy safe to use in a task.

Alpaca: Tasks with Dynamic Privatization

Alpaca tasks are restartable *without* a checkpoint of volatile state

---

**Task**

Compiler adds code to *automatically commit* copied state back to main memory

---

[Maeng, et al OOPSLA '17]
origin task Sense()
  
  S=read_sensor()

NextTask CmpAvg }

Shared S, Cnt, A[5], head

Function-as-task means no need to checkpoint stack & regs

Alpaca: Tasks with Dynamic Privatization

[Maeng, et al OOPSLA '17]
for( i = 1)
call append()
buf[sz] = 'a'
sz++

task_boundary

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

Correctly Updated Only Once!

Tasks Eliminate Surprising Behavior
int NV_Array[1000000];

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

task_boundary

Key Question:
Which data to selectively privatize?
int NV_Array[1000000];
task_boundary
i = rand(1000000);
NV_Array[i]++;
task_boundary

Which task-shared data to privatize?
Data written before a failure that may be read after reboot
Prototype Implementations

DINO, Chain, Alpaca

Task-aware Programming Model

Task-aware Compiler

Runtime Library

Task Data-flow Analysis
Link to Task Runtime
Channels, Versioning, Privatization

DINO: Checkpoint & Recovery
Chain: Task Management
Alpaca: Privatization/Commit
Energy-harvesting Platform Prototypes

Applications
Activity Recognition (accelerometer+ML)
Deep Neural Networks (MNIST, OKGoogle, Activity Recognition)
[ASPLOS’19]

Cold-chain Equipment Monitor
Multi-granular Sensor Log

Key Result:
Applications suffer errors & failures without our system support for tasks
Alpaca outperforms DINO and Chain

Normalized Run Time

- Alpaca
- Chain
- DINO

CEM
AR
RSA
CF
BF
BC

Alpaca outperforms DINO and Chain
Chinchilla

[OSDI '18]

Efficient Automatic Checkpoints without Non-termination
Chinchilla Supports Safe, Efficient Checkpoint-based Intermittent Execution

```
\begin{align*}
&i := 0 \\
&\text{sum} := 0 \\
&i < A\_LEN\? \\
&\text{sum} := \text{sum} + A[i] \\
&i := i + 1
\end{align*}
```

Non-volatile
Too Few Checkpoints May Cause Non-termination

i := 0
sum := 0

i < A_LEN?

sum := sum + A[i]
i := i + 1

... Cannot reach the next checkpoint
→ Non-termination
Too Many Checkpoints Incur High Overhead

High overhead!
Checkpoint Placement Determines Efficiency and Correctness

Non-terminating!

High overhead!

\[ i := 0 \]
\[ \text{sum} := 0 \]

\[ i < \text{A_LEN} ? \]

\[ \text{sum} := \text{sum} + \text{A}[i] \]
\[ i := i + 1 \]

\[ i := 0 \]
\[ \text{sum} := 0 \]

\[ i < \text{A_LEN} ? \]

\[ \text{sum} := \text{sum} + \text{A}[i] \]
\[ i := i + 1 \]
Requirements for Checkpoint Insertion

- Frequent
- Low energy variance
- Compiler-friendly

Basic Block checkpoints avoid non-termination if no block exceeds device energy
Chinchilla Checks Block Energy to Avoid Non-termination

Energy of an 47uF (small) Capacitor

- CEM
- CF
- RSA
- AR
- BF
- BC

Energy (uJ)

Min | Avg | Max
Block Energy is Far Below Device Energy

Chinchilla’s Block Checkpoints Avoid Non-termination

Energy of an 47uF Capacitor

21x headroom
Checkpoint Placement Determines Efficiency and Correctness

Non-terminating! Safe!

High overhead!

```
i := 0
sum := 0
```

```
i < A_LEN?
```

```
sum := sum + A[i]
i := i + 1
```

...
Checkpoint Placement Determines Efficiency and Correctness

Non-terminating!

High overhead!

```
i := 0
sum := 0

i < A_LEN?

sum := sum + A[i]
i := i + 1
...
```
Chinchilla Selectively Checkpoints to Avoid High Overheads

• Key Idea:
  – Checkpointing on each block is **worst case** reasoning.
  
  – **Average case**: sufficient energy for multiple blocks to complete.

  – Chinchilla’s idea: **selectively skip** checkpoints to avoid high cost.

  – **Gracefully Degrade** to worst case checkpoints if necessary.
Chinchilla Uses Timer-Based Selective Checkpointing

\[
\begin{align*}
T \\
& \text{\textbf{Ti:=0}} \\
& \text{\textbf{sum:=0}} \\
& \text{\textbf{i < A_LEN?}} \\
& \text{\textbf{sum:=sum+A[i]}} \\
& \text{\textbf{i:=i+1}} \\
\end{align*}
\]
Chinchilla Uses Timer-Based Selective Checkpointing

i := 0
sum := 0

i < A_LEN?

sum := sum + A[i]
i := i + 1
Chinchilla Uses Timer-Based Selective Checkpointing

```
i := 0
sum := 0
```

```
i < A_LEN?
```

```
sum := sum + A[i]
i := i + 1
```

...
Chinchilla Uses Timer-Based Selective Checkpointing

\begin{align*}
i &:= 0 \\
\text{sum} &:= 0 \\
\text{Skip} \\
i &< A\_LEN? \\
\text{sum} &:= \text{sum} + A[i] \\
i &:= i + 1 \\
\end{align*}
Chinchilla Uses Timer-Based Selective Checkpointing

\begin{align*}
  i & := 0 \\
  \text{sum} & := 0 \\
  \text{if} & \quad i < A_{\text{LEN}}? \\
  \text{then} & \quad \text{sum} := \text{sum} + A[i] \\
  \quad i & := i + 1 \\
  \text{else} & \quad \text{Skip} \\
  \text{end} \end{align*}
Chinchilla Uses Timer-Based Selective Checkpointing

i:=0
sum:=0

i < A_LEN?

sum:=sum+A[i]
i:=i+1

...
Chinchilla Uses Timer-Based Selective Checkpointing

\[
i := 0 \\
\text{sum} := 0
\]

\[
i < \text{A_LEN}?
\]

\[
\text{sum} := \text{sum} + A[i] \\
i := i + 1
\]

Checkpoint!
Chinchilla Adapts its Checkpointing Interval

- Dynamically find the right checkpointing interval during execution by **binary search**
- Automatically **adjusted** on environmental changes
Checkpoint Placement Determines Efficiency and Correctness

Non-terminating! Safe!

High overhead! Fast!

```
i:=0
sum:=0

i < A_LEN?

sum:=sum+A[i]
i:=i+1

...```

Electrical & Computer Engineering
Chinchilla Is Safe and Efficient

Non-termination if Alpaca task too large

Chinchilla Is Safe and Efficient

Brandon Lucia - Carnegie Mellon University - Challenges of Intermittent Computing
Use Chinchilla, Alpaca, Chain and DINO for your research!

http://intermittent.systems | http://github.com/CMUAbstract

Alpaca
A programming language for writing reliable software for intermittent, energy-harvesting computer systems.

Download VM (10.67GB)
VM contains Alpaca and other previous state-of-the-art systems (Chain, DINO)
UserID: reviewer PW: review.

Pull source code from Github
https://github.com/CMUAbstract/alpaca-oopsla2017
Intermittent & Physically Constrained Computing Platforms
Hardware, Programming Models, System Software
Key Insight: Co-design PL, OS, Arch, and power system for intermittent operation

Intermittence-safe HW/SW Systems [ASPLOS 2018]

EDBSat Chip-scale satellite
- 3cmx3cmx4mm, solar power
- Magnetometer + gyro sensors
- Low-power radio (to earth)
- Custom “burst” power system
- Integrated HW/SW diagnostics
- Programmed in intermittence-safe Chain programming lang.
- Launch partner: KickSat/NASA

Capybara multi-sensor platform
- 5cmx5cm, solar/RF power
- 10 sensor array
- Bluetooth Low-energy Radio
- Software-switchable variable-capacity “burst” power system
- Integrated HW/SW diagnostics
- Supports Alpaca & Chain intermittent programming langs.
EDBSat is in LEO now!

KickSat-2 Mission w/ Zac Manchester (Stanford)
Capybara’s Co-designed HW & SW Meet Dynamically Varying Energy Demand

Configurable HW Energy Storage

Programmable Task Energy Modes

- Reactivity constraint: $T_{off} < T_{sample}$
- Atomicity constraint: $T_{on} > T_{tx}$
- Asynchronous burst constraint

Brandon Lucia - Carnegie Mellon University - Challenges of Intermittent Computing
EDB: The Energy-interference-free Debugging Toolchain
[Colin, et al. CASES '15, ASPLOS '16, IEEE Micro Top Picks '16]

Energy-interference-free debugging and monitoring for intermittent, energy-harvesting computers.

Simultaneously monitor & manipulate program state and energy state.
Intelligence beyond the Edge

Deep neural network inference optimized for intermittent operation

*First demonstration of DNN inference on an intermittent device*

Challenge: Resource constrained & intermittent

Software-Only Neural Intermittent Computing (SONIC) & TI Accelerated Intermittent Linear algebra Solver (TAILS)
Jetson TX2 Compute Module

Capacitor-based energy storage

3U CubeSat Nanosatellite

Orbital Edge Computing: Machine inference in camera-equipped computational nanosatellites

[GoMACTech’19, CAL’19]
Orbital Edge Computing

Distributed platform designs for high-performance orbital edge computing

[GoMACTech’19, CAL’19]
Edge Graph Processing

Single-machine processing of network structured data.

[IISWC’18 Best Paper, HPDC ‘19 (in sub)]
Solving the System Challenges of Intermittent Energy-harvesting Devices

Brandon Lucia | Assistant Professor, CMU ECE

ABSTRACT Research Group - https://intermittent.systems

with PhD Students: Kiwan Maeng, Emily Ruppel, Vignesh Balaji
Graham Gobieski, Milijana Surbatovich, Brad Denby, Harsh Desai

PhD Alumni: Alexei Colin,

Industry Collaborators: Zac Manchester (Harvard/KickSat)
Alanson Sample (Disney Research Pittsburgh)