Intermittent Energy-Harvesting Computing Systems

ML/AI beyond the edge [Sonic & Tails, ASPLOS 2019]

[Capybara, ASPLOS 2018]
Intermittent devices need concurrent interrupts

Atomic w.r.t power failures

Not intermittence safe

[Alpaca, OOPSLA 2017]

[Chain, OOPSLA 2016]
Intermittent devices need concurrent interrupts

- Event-atomic tasks
- Multi-task transactions
- Timely split-phase events
Coati Motivation

- Intermittent Programming Models & Interrupts
- Problem #1: Interrupt Interrupted
- Problem #2: False Flag

Coati System Design

- Coati Overview
- Task & Event Interaction
- Multi-Task Transactions

Coati Evaluation

- Correctness
- Programming Effort
- Runtime Overhead
Prior work supports one execution context

\[ \text{assert}(X == Y) \]

next_task send()

\[ W = Z \]
\[ Y++ \]
\[ \text{assert}(X == Y) \]

[Alpaca, OOPSLA 2017]

\[ Z = X \]
\[ X++ \]

\[ W = Z \]
\[ Y++ \]
\[ \text{assert}(X == Y) \]

idempotent re-execution

write-after-read dependence

task incX()
\[ Z = X \]
\[ X++ \]

next_task incY()

task incY()
\[ W = Z \]
\[ Y++ \]
\[ \text{assert}(X == Y) \]

next_task send()
Interrupts violate correctness assumptions

Assumptions:
• Tasks always start from the same point
• Tasks re-execute idempotently

[Alpaca, OOPSLA 2017]
Problem #1: Interrupt Interrupted
Power failures can happen during interrupts

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

task increment()

```
...  
t1 = X + 1
...
  t2 = Y + 1
assert(t1==t2)
  next_task send()
```

event clear()

```
  X = 0
  Y = 0
  return
```
Problem #2: False Flag

Conventional synchronization fails

task incX()

...  
    flag = 1  
    T1 = X + 1  
next_task incY()

task incY()

...  
    T2 = Y + 1  
    assert (T1==T2)  
    flag = 0  
next_task send()

event clear()

if(flag) return
    X = 0
    Y = 0
return

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>X</th>
<th>Y</th>
<th>flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Coati Motivation

- Intermittent Programming Models & Interrupts
- Problem #1: Interrupt Interrupted
- Problem #2: False Flag

Coati System Design

- Coati Overview
- Task & Event Interaction
- Multi-Task Transactions

Coati Evaluation

- Correctness
- Programming Effort
- Runtime Overhead
Coati produces correct code with interrupts for intermittent systems
Coati serializes interrupts after scheduled tasks

Atomic w.r.t power failures & events
Atomic w.r.t events not power failures

(transaction

\textbf{task} \texttt{incX()}

\begin{itemize}
  \item \texttt{X++}
  \item \texttt{next\_task incY()}
\end{itemize}

\textbf{event sensor()}

\begin{itemize}
  \item \texttt{x = sense()}
  \item \texttt{event\_return()}
\end{itemize}

\textbf{task incY()}

\begin{itemize}
  \item \texttt{Y++}
  \item \texttt{assert(x == y)}
  \item \texttt{next\_task send()}
\end{itemize}
Split-phase events provide timely reaction

\begin{align*}
\text{event} & \quad \text{sensor}() \\
X & = \text{senseX}() \\
Y & = \text{senseY}() \\
\text{Avg}X & = \text{movingAvg}(X) \\
\text{Avg}Y & = \text{movingAvg}(Y) \\
\text{evt\_return}() \\
\end{align*}

\begin{align*}
\text{top} & \quad \text{sensor}() \\
X' & = \text{senseX}() \\
Y' & = \text{senseY}() \\
\text{top\_return}() \\
\end{align*}

\begin{align*}
\text{bottom} & \quad \text{sensor}() \\
\text{Avg}X & = \text{movingAvg}(X') \\
\text{Avg}Y & = \text{movingAvg}(Y')
\end{align*}
Split-phase serialization orders tasks & events

**task** increment()

...  
X++
...  
Y++
assert(X==Y)

**next_task** send()

**top** sensor()

X' = senseX()
Y' = senseY()
evtx_return(sensor)

**bottom** sensor()

AvgX = movingAvg(X')
AvgY = movingAvg(Y')

**task** send()
Coati serializes multiple interrupts in FIFO order

task incX()

\[ X++ \]

next_task send()

task incY()

\[ Y++ \]

assert (X==Y)

tx_next send()

bottom sensor()

\[ \ldots \]

top sensor()

\[ \ldots \]

evt_return(sensor)

bottom sensor()

\[ \ldots \]

bottom sensor()

\[ \ldots \]

evt_return(sensor)

bottom sensor()

\[ \ldots \]

task send()
Coati Motivation

- Intermittent Programming Models & Interrupts
- Problem #1: Interrupt Interrupted
- Problem #2: False Flag

Coati System Design

- Coati Overview
- Task & Event Interaction
- Multi-Task Transactions

Coati Evaluation

- Correctness
- Programming Effort
- Runtime Overhead
Coati Evaluation

• Correctness
• Programming Effort
• Runtime Overhead
We evaluated Coati on real hardware

<table>
<thead>
<tr>
<th>Systems</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coati</td>
<td>Bitcount</td>
</tr>
<tr>
<td>Alpaca</td>
<td>Activity Recognition</td>
</tr>
<tr>
<td>Atomic</td>
<td>RSA Encryption</td>
</tr>
<tr>
<td>Hand-Optimized</td>
<td>Cold-Chain Equipment Monitor</td>
</tr>
<tr>
<td></td>
<td>Cuckoo Filter</td>
</tr>
<tr>
<td></td>
<td>Blowfish Encryption</td>
</tr>
</tbody>
</table>
Coati ensures correct execution with interrupts

- Synchronization was carefully added to the Alpaca code
- Both systems used the same task decomposition
- Data in Alpaca still become inconsistent

<table>
<thead>
<tr>
<th></th>
<th>Coati</th>
<th>Alpaca</th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AR</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>RSA</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>CEM</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>CF</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>BF</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>
Coati transactions simplify concurrent code

- Lines of synchronization code reduced by 67% on average
- On average, fewer, simpler transitions are required
Coati’s overhead is similar to other strategies without their drawbacks.
Coati provides simple, correct semantics for concurrency control in an intermittent execution

For more:
• Read the paper
• Experiment with our code
• Stick around!

github.com/CMUAbstract/coati_pldi19.git
Transactional Concurrency Control for Intermittent, Energy-Harvesting Computing Systems

Emily Ruppel and Brandon Lucia
June 26, 2019
PLDI 2019 -- Systems II