PROGRAM ENERGY HOTSPOT DETECTION AND REMOVAL: A STATIC ANALYSIS APPROACH

Presenter: Mohsen Shekarisaz
Authors: Mohsen Shekarisaz, Fatemeh Talebian, Marjan Jabariani, Farzad Mehri, Fathiyeh Faghih, Mehdi Kargahi
I received my B.Sc. in computer engineering from University of Isfahan and my M.Sc. from Sharif University of Technology in 2015 and 2017, respectively. I am currently a Ph.D. student in the school of electrical and computer engineering, University of Tehran. My research interests include energy testing, energy bugs and hotspots, memory management in non-volatile architectures, and low-power real-time embedded systems.
OUTLINE

- Introduction
- Related studies
- System model and problem statement
- Proposed approach
- Evaluation
- Conclusion
INTRODUCTION

- Most embedded devices in the context of Cyber-Physical Systems (CPS) and Internet-of-Things (IoT) are battery operated
  - Significant role of embedded software on the way that energy is consumed (Chemeris et al., 2010)

- Various factors contribute to the energy consumption of an embedded software
  - Computation parts
  - Energy hungry external resource modules (e.g. LTE, WiFi, GPS) (Mittal et al., 2012)
    - Registering/de-registering by acquire/release statements

- Energy issues: Inefficient usage of resources in software (Banerjee et al., 2014)
  - Energy bugs (e.g. resource leak)
  - Energy hotspots (e.g. inefficient resource usage or energy wastage in resource)
INTRODUCTION

- Revealing energy issues through static analysis in previous studies
  - Analyzing the program statically, usually based on the control flow graph (CFG) of the code
  - Frameworks to detect, validate and repair resource leak (a type of energy bugs)

- Challenge with revealing some of the energy hotspots
  - They are not easily exposed to the program inspection
  - Example: Hotspot due to inappropriate releasing/acquiring the resource
What did we do in this research?

- Defining two new hotspots based on the mode switches (acquire and release operations)
  - Hotspots based on the locations in the software code
  - Using Best-Case Execution-Time (BCET) and Worst-Case Execution-Time (WCET) of specific program segments (Called partial BCET/WCET)

- Investigating the program code statically to detect these types of energy hotspots and repairs them
  - A static analysis approach, performed on Control Flow Graph (CFG) of the program

- Evaluating our approach by experiments performed on an embedded system
**RELATED STUDIES**

- **Energy testing to investigate the existence of energy bugs/hotspots**
  - (Banerjee et al., 2014), (Jabbarvand et al., 2019)
  - The program needs to be executed by test inputs to find energy issues

- **Static analysis to find energy anti-patterns like resource leaks**
  - (Banerjee et al., 2017), (Jiang et al., 2017)
  - Restructuring the code without changing its behavior to reduce energy consumption
  - (Chen et al., 2017), (Palomba et al., 2018)
  - The focus of them has been on revealing energy bugs
  - Two energy hotspots patterns have not been considered
  - Relation between extreme scenarios (WCET/BCET) and resource power mode switching has not been considered
A computation unit, called CPU, and a single module, called resource

Resource r

<table>
<thead>
<tr>
<th>Resource Modes</th>
<th>Description</th>
<th>Switching Overheads</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active mode</td>
<td>by Acquire(r)</td>
<td>$D_{Acq}$ and $E_{Acq}$</td>
<td>$\text{Power}_A(r)$</td>
</tr>
<tr>
<td>Power Saving mode</td>
<td>by Release(r)</td>
<td>$D_{Rel}$ and $E_{Rel}$</td>
<td>$\text{Power}_S(r)$</td>
</tr>
</tbody>
</table>

CPU States

<table>
<thead>
<tr>
<th>CPU States</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle state</td>
<td>CPU waits for $r$ to become active</td>
</tr>
<tr>
<td>Busy state</td>
<td>CPU runs computational instructions</td>
</tr>
</tbody>
</table>
Program P is represented by a graph (CFG)

Four Types of Nodes

- **Acquire** Node: A node with a single Acquire(r)
- **Release** Node: A node with a single Release(r)
- **Use** Node: A node with one or more Use(r) statements
- **Ordinary** Node: A node with only computation statements (i.e. with no Acquire(r), Release(r), or Use(r))

Functions for node b and set of nodes B

- \( B = next_{\text{Acq}}(b) \): Set of immediate Acquire Nodes in the paths of the partial graph starting at b
- \( \text{Similar for} \ next_{\text{Rel}}(b) \ , \ next_{\text{Use}}(b) \)

- **PartialWCET**(b, B) = The worst-case execution time from b to an immediate Acquire(r) with the most distance
- \( \text{Similar for} \ \text{PartialBCET}(b, B) \)
1. Acquire(r);
2. int i=0;
3. if(j>0){
4.     j++;
5.     Use(r);
6.     Release(r);
7. }else{
8.     Use(r);
9.     i=j-1;
10.    j++;
11.    Release(r);
12. }

A sample code

A sample modified CFG
PROBLEM STATEMENT

- Assumptions about functionally correct program P
  - Neither Acquire(r) nor Release(r) do conflict with the computational statements
  - Sequence C-Acquire(r)-Use(r)...Use(r)-C-Release(r)-C must be preserved
    - C represents zero or more computational statements
    - Multiple occurrences of the sequence are possible

- Two operations on P related to resource r
  - Removing a statement of Release(r) and the subsequent statement of Acquire(r)
  - Moving the statement of Acquire(r) up or down while there is no conflict

- Goal
  - Using the above operations towards getting program P” with better (or at least same) energy consumption of each individual path in compare to that of P
Two energy hotspot patterns with regards to energy related questions:

- **Energy hotspot Type 1 (EHT1):** For any two subsequent pairs of Acquire(r)-Release(r), is it better to remove the first Release(r) and the second Acquire(r), so that the two pairs will be converted to one Acquire(r)-Release(r) pair?

  ![EHT1 Diagram]

- **Energy hotspot Type 2 (EHT2):** For any pair of Acquire(r)-Use(r), is Acquire(r) placed in its optimal location with respect to the first Use(r) after it?

  ![EHT2 Diagram]
The general procedure: Hotspot Removal (HR) algorithm

```
Algorithm 1: HR Algorithm

Input : Program $P$, Resource Specification $S$
Output: Energy-Improved Program $P''$
1: $P' \leftarrow \text{removeEHT1}(P, S)$;
2: $P'' \leftarrow \text{removeEHT2}(P', S)$;
3: return $P''$;
```
DETECTION AND REMOVAL

- Detection and Removal of EHT1 algorithm
  - Lower Bound on Sleep Time (LBST): Minimum time that the resource needs to be in sleep mode in order to eliminate the Release(r) and Acquire(r) energy overheads

\[
LBST = \frac{E_{Rel} + E_{Acq} - Power_{S}(r) \times (D_{Rel} + D_{Acq})}{Power_{A}(r) - Power_{S}(r)}
\]
DETECTION AND REMOVAL

Algorithm 2: EHT1 Algorithm

Input: Program P, Resource Specification S
Output: Energy-Improved Program P'
1: $MCFG \gets \text{CreateModifiedCFG}(P')$
2: $LBST \gets \text{calculateLBST}(S)$
3: $RN \gets MCFG\text{.getAllReleaseNodes}()$
4: for all $rn \in RN$ do
5: $AN_{rn} \gets MCFG\text{.getAllNextAcquireNodes}(rn)$
6: $WCET \gets \text{CalculatePartialWCET}(rn, AN_{rn})$
7: if $WCET < LBST$ then
8: $MCFG\text{.remove}(rn)$
9: $MCFG\text{.remove}(AN_{rn})$
10: if there are two release nodes $rn'$ and $rn''$ without an acquire node in between then
11: $n \gets MCFG\text{.getNextPotentialAcquireNode}(rn')$
12: $WCET \gets \text{CalculatePartialWCET}(rn', n)$
13: if $WCET < LBST$ then
14: $MCFG\text{.remove}(rn')$
15: else
16: $an \gets \text{newAcquireNode}$
17: $MCFG\text{.add}(an, n)$
18: end if
19: end if
20: end if
21: end for
22: $P' \gets \text{buildEnergyImprovedProgram}(MCFG)$
23: return $P'$;

- Detection and Removal of EHT1 algorithm
DETECTION AND REMOVAL

Algorithm 3: EHT2 Algorithm

Input : Program $P'$, Resource Specification $S$
Output : Energy-Improved Program $P''$

1: $MCFG \leftarrow \text{CreateModifiedCFG}(P')$;
2: $AN \leftarrow MCFG.\text{getAllAcquireNodes}()$;
3: for all $an \in AN$ do
4: $UN_{an} \leftarrow MCFG.\text{getAllNextUseNodes}(an)$;
5: $BCET \leftarrow \text{CalculatePartialBCET}(an, UN_{an})$;
6: if $BCET < D_{Acq}$ then
7: $\text{OptimalPoint} \leftarrow MCFG.\text{findOptimalPoint}(an)$;
8: $MCFG.\text{moveBackNode}(an, \text{OptimalPoint})$;
9: end if
10: end for
11: $P'' \leftarrow \text{buildEnergyImprovedProgram}(MCFG)$;
12: return $P''$. 

Detection and Removal of EHT2 algorithm
EVALUATION: SETUP AND EXPERIMENTS

Setup
- Arduino Uno with an ATmega328P microcontroller
- WiFi module ESP8266
- USB-4716 Advantech Data Logger for measuring energy consumption
- Bound-T tool for evaluating WCET/BCET of the program segments
- Malardalen benchmark suite as the embedded programs

Experiments
- LBST = 3.16 s
- \( D_{\text{Acq}} = 2.945673333 \text{ s} \), \( D_{\text{Rel}} = 0.010436667 \text{ s} \)
- \( E_{\text{Acq}} = 16.274765 \text{ mJ} \), \( E_{\text{Rel}} = 0.469583 \text{ mJ} \)
- \( \text{Power}_{A} = 5.31213 \text{ mW} \), \( \text{Power}_{S} = 0.3902 \text{ mW} \)
EVALUATION: EMBEDDED PROGRAM

Program P()
{
    Comp_1();
    Acquire(r);
    Use(r);
    Release(r);
    Comp_2();
    Acquire(r);
    Use(r);
    Release(r);
    ...
    Comp_n();
    Acquire(r);
    Use(r);
    Release(r);
}

- The program structure
**EVALUATION: RESULTS**

- The effect of removing each Release/Acquire pair on energy

  - **Removing EHT1**: The total energy consumption is reduced from 150.51 mJ to 118.88 mJ (about 21%)
  - **Removing EHT2**: the total energy consumption is reduced from 118.88 mJ to 108.36 mJ (about 8.8%)
  - **HR Algorithm** reduces the total energy from 150.51 mJ to 108.36 mJ (about 28%)

- The process of energy reduction by the HR algorithm
EVALUATION: RESULTS

- Two important facts
  - The number of release/acquire operation pairs are reduced (based on EHT1), so the energy and time overheads of acquire/release operations.
  - The idle times of CPU imposed by the acquire operations are eliminated (based on EHT2), which decrease the active power consumed by CPU.

- The power traces of the original and energy-improved programs
CONCLUSION

- Importance of non-functional aspects, including energy consumption, in embedded systems
- Energy hotspots in program due to inefficient use of peripheral modules
- **Goal of this research:** A code level static analysis method for energy hotspot detection
  - Based on some relation between WCET/BCET of some program segments and time/energy specifications of the module
- **Experiments:** Up to 28 percent of energy reduction after the hotspot removals
- **Future work:** Extending the study by more patterns and analysis methods
REFERENCES


THANKS FOR YOUR ATTENTION