Estimate Energy-Use of Mobile Apps

- Building energy-efficient applications will increase the user experience
  - How much energy is consumed?
  - Where in the code is more energy consumed?

- Possible approaches may use:
  - Specialized hardware
  - Cycle-accurate simulators
  - OS instrumentation

ELens

- Combine “program analysis” and “per-instruction energy modeling” technique to obtain fine-grained estimates of application energy.
  - No specialized hardware
  - No cycle-accurate simulator
  - No OS instrumentation

- Approach
  - Instrument application code
  - Monitor runtime information
  - Estimate energy consumption at various granularities
Approach: Workload Generation

- **Workload describes use cases**
  - Instrument the source code.
  - Run use cases against the instrumented code.
  - Record paths traversed.
    - Sequence of (method, path) pairs
      \[ \{(a,1), (b,1), (a,2), (a,3)\} \]
  - Track hardware usage
    - On/Off (e.g., Wi-Fi)
    - CPU frequency
    - ...

Approach: Est. Energy Use per Instruction

- **Estimate energy use at various granularity levels**
  - Generate complete instruction sequence for each recorded path
  - For each path, produce mapping between instruction and method, and between instruction and line number
  - Estimate cost for each inst.
    - Consider data volume processed.
    - Consider hardware state when the inst. is under execution.
  - Return the sum.

Approach: Est. Energy Use per Instruction

- **Algorithm**
  
  **Input:** $H$: set of hardware components, $l$: source line, $m$: method, and $P$: path
  
  **Output:** Energy estimate in Joules
  
  1: \[ \text{cost} \leftarrow 0 \]
  2: \[ (\Theta, M, L) \leftarrow \text{regenerate}(P) \]
  3: \[ D \leftarrow \text{propagateDT}(\Theta) \]
  4: for all $h \in H$
  5:     for all $i \in \Theta$
  6:         $f_h \leftarrow \text{powerstate}(i, h)$
  7:         if $L(i) = L \wedge M(i) = m$ then
  8:             $\text{cost}(h) \leftarrow C(i, h, f_h, D(i))$
  9: return $\sum_{h \in H} \text{cost}(h)$

Approach: Energy Annotation

- **Generate feedback for developer**
  - Convert path information and energy estimation into a graphical representation.
    - Whole application
    - Per method, source code line, path
  - Show estimated energy in colors in Eclipse.
How Much Energy for Instruction?

- Software Environment Energy Profile (SEEP)
  - Provides per-instruction energy cost functions for each component of the target platform
  - But, not common yet in practice. (manufacturers have to provide)

- Used Low-Energy Aware Platform (LEAP) power measurement device.
  - Measure energy use per instruction type on the LEAP platform.

Experimental Setup and Results

- Preprocessing
  - Convert Dalvik bytecode (subject application) to Java bytecode
  - Instrument the Java bytecode using a library
  - Convert the instrumented Java bytecode to Dalvik bytecode
  - Deploy the Dalvik bytecode to the LEAP platform
- Actual energy consumption can be obtained by running app on the LEAP platform.

Java Bytecode vs. Dalvik Bytecode

- Architecture

Source: http://forensics.expertm術inart.de/2012/08/27/comparison-of-dalvik-and-java-bytecode/

Results

- Accuracy
  - Compare with the eLens estimate with the actual use.
  - Average bytecode case: Uniform energy for all instructions
  - No path sensitivity: Ignore path-sensitive cost
Other Findings

- Correlation between the energy use and the execution time of a method is low.
- eLens can be used to study energy hotspots.

Energy Use Patterns for Android Apps

- Experimental setup
  - Energy consumption estimate by vLens
  - 405 real-world Android apps
    - Exclude games because complex and nondeterministic operations are needed
    - Monsoon power meter for measuring the power consumption over time
    - Monkey for generating pseudo random stream of user inputs and events for applications
  - Run on controller desktop and push events to smartphone through the network
  - 5 UI events per second and 500 events in total
  - Completely automated

Energy Consumption by Each App

- Measure:
  - Total energy consumed by each app during execution

An Empirical Study of the Energy Consumption of Android Applications

Energy Consumption in Idle State

- **Measure:**
  - API-Idle
  - Energy consumed while sleeping APIs
  - Non-Idle
  - Energy consumed by the app for executing all paths, excluding the API-Idle energy
  - Pure-Idle
  - Energy consumed while no app code is running
  - Total energy minus API-Idle and Non-Idle.

- **Findings**
  - App code explains less than 31% of the total energy! Hence, simply optimizing the energy consumption of user code is not sufficient.

Then, where was the energy used?

Which Code Eat Up the Energy the Most?

- **Comparison between system API and developer code**
  - **Measure:**
    - Divide Non-Idle into three categories
      - API – energy of invoking APIs in Android SDK
      - Bytecodes – energy consumption by normal use code
      - Outliers – energy introduced by system events

System API uses dominate non-Idle time energy use for most apps!

Energy Consumption per H/W Components

- **Measure:**
  - Ratio of energy consumed by eight hardware components to the total non-idle energy consumption
  - Considered APIs relevant to control the components
  - For the average, excluded apps never used a certain component

Network is the most energy consuming component!

Socket < 1%

WebKit 10%

High Energy Consuming APIs

- **How many APIs are significant in energy consumption?**
  - Measure the ratio of energy for each API over all non-idle energy across all apps.
  - Measure the ratio of energy for top ten most energy-consuming APIs over all non-idle energy in each app.

\[
\text{Ratio}_i = \frac{\sum_{j=1}^{N} R_{ij}}{N} \quad \text{Top10}_i = \frac{\sum_{j=1}^{10} E_{ij}}{E_i}
\]

98% of APIs have ratio below 0.1%

Only a few APIs are significant in energy consumption
High Energy Consuming APIs
- How similar are the top ten most energy consuming APIs across apps?
  - Measure the overlaps for all app pairs
  - Average and median is close to 1.
- Which APIs are most likely to be in the top ten?
  - 5 APIs are in top 10 in more than 100 apps.
    - 4 are related to HTTP requests
    - 1 is related to file synchronization

Energy Consumption by Code in Loop
- Cases
  - Loops with HTTP request
  - Loops with other APIs
  - Loops with no API
- Loops with HTTP request consumes significant amount of energy!

Energy Consumption by Bytecodes
- Measure:
  - Energy consumption of each bytecode to the total non-idle energy consumption of applications.
- Data manipulation code is invoked more frequently.

Time = Energy?
- Measure for each app:
  - Top 10 most energy consuming APIs
  - Top 10 most time consuming APIs
- Map the ranking of the time consuming APIs in descending order to that of the energy consuming APIs in descending order. Check if APIs are ranked same.
- Ranked the same ~ 4.6. stdev ~ 2. Ranking varies.
Sufficient Measurement Granularity

- Nanosecond level vs. Millisecond level.
- Compute: $\frac{\text{milli}}{\text{nano}}$
  - \text{milli} and \text{nano} are the two measurements.
- Mean error for non-idle energy is \(~64\%\).
  - Max was \(~2,500\%).

Coarse-grained measurement in milliseconds are not sufficient.

Account for Idle State Energy

- System consumes energy while an app is in the idle state.
- Measure:
  - IdleKept
    - Only total energy consumption of an app – sum up all energy samples during the app execution
  - IdleSubtract
    - Energy consumption during non-idle state.
- Average difference between IdleKept and IdleSubtract is 36%.

Measuring total energy consumption only is inaccurate and can mislead results.

Wrap Up

- Energy Consumption Patterns
  - Apps consumed over 60% of their total energy in idle states.
  - Network is the most energy consuming component
  - A few system APIs dominate the energy consumption.
  - Data manipulation operations are the most energy consuming.

- Analysis of common practices
  - Time as an approximation of energy consumption, coarse-grained measurement, and the negligence of idle energy can mislead the developers to spend their efforts on incorrect optimizations.