
Xueliang Li and John Gallagher

Roskilde University

ICT-Energy Science Conference
Aalborg
17-19 August 2016

Supported by
EU FP7 Project ENTRA:Whole-Systems Energy Transparency
EU FP7 CSA ICT-Energy
Energy-aware mobile app development

- Energy efficiency of apps is an important design goal.
  - Many interactive apps for mobile devices (e.g. games) require high performance for fast response
    → higher CPU clock speed
  - Frequent charging frustrates users
Tools for energy-aware development

• An energy-aware software developer needs to understand quickly the energy profile of code
  
  – how much energy does the app consume?
  – which part(s) of the code cause the most energy consumption?
Energy-awareness during development

- Measuring energy efficiency on hardware, after the software is developed, is not an option.

- It might be too late to fix “energy bugs”
Energy transparency

Energy is consumed by physical processes.

Can programmers “see” through the layers and understand energy consumption at the level of source code?
One approach—mapping low-level model
Energy transparency for Android game code energy optimisation

• Experiment by Xueliang Li, Roskilde University
• Energy of game code is highly dependent on user interaction
• We aimed to model the energy of source code operations in the Cocos2d-Android game engine
Energy transparency for app code

A mapping from low-level is very complex, probably not feasible.
Modelling and optimisation framework

Data Collection
- Raw Energy Data
  - Running on Execution Cases
    - Application Code
  
Model Construction
- Regression Analysis
  - Operation Based Model
    - Energy Accounting
      - Energy Profiles

Energy Accounting
- Energy Accounting
  - Refactoring Strategy Set

Code Optimization
- Hot Spots Identification
  - Strategy Selection
    - Code Refactoring
      - Optimised Code

Understanding Energy Features
Typical game use-cases

Click and move

3D effect

Input sequence 1: (tap, position1), (tap, position2)
Input sequence 2: (tap, position1), (tap, position2)
Input sequence 3: (tap, position1), (tap, position2)

Input sequence n: (tap, position1), (tap, position2)
Energy measurement of test cases

\[ n_j^{(i)} = \text{# executions of op}_j \text{ in case } i \]
Test scenarios

• Typical game interaction scenarios
  – Click and move
  – Orbit
  – Wave
• For each scenario, a large number of tests was generated
• Some random permutation and deletion of code to introduce more variation in tests
Power measurement setup

- Odroid-XU+E development board
- two ARM quad-core CPUs
  - Cortex-A15 at 2.0 GHz
  - Cortex-A7 at 1.5 GHz
- built-in sensors for collecting CPU power samples during test runs
- sampling rate 30 Hz
- sum of power samples to yield total energy of each test

\[ E = \int_{t_0}^{t_n} \text{power}(t) \, dt \approx \sum_{i=1}^{n} \text{power}(t_i) \cdot \Delta_i \]

where \( t_0 \leq t_1 \leq t_2 \cdots \leq t_{n-1} \leq t_n \)
Examples of source energy ops

<table>
<thead>
<tr>
<th>Operation</th>
<th>Identified where:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method Invocation</td>
<td>one method is called</td>
</tr>
<tr>
<td>Parameter_Object</td>
<td>Object is one parameter of the method</td>
</tr>
<tr>
<td>Return_Object</td>
<td>the method returns an Object</td>
</tr>
<tr>
<td>Addition_int_int</td>
<td>addition’s operands are integers</td>
</tr>
<tr>
<td>Multi_float_float</td>
<td>multiplication’s operands are floats</td>
</tr>
<tr>
<td>Increment</td>
<td>symbol &quot;++&quot; appears in code</td>
</tr>
<tr>
<td>And</td>
<td>symbol &quot;&amp;&quot; appears in code</td>
</tr>
<tr>
<td>Less_int_float</td>
<td>&quot;&lt;&quot;’s operands are integer and float</td>
</tr>
<tr>
<td>Equal_Object_null</td>
<td>&quot;==&quot; ’s operands are Object and null</td>
</tr>
<tr>
<td>Declaration_int</td>
<td>one integer is declared</td>
</tr>
<tr>
<td>Assign_Object_null</td>
<td>assignment’s operands are Object and null</td>
</tr>
<tr>
<td>Assign_char[]_char[]</td>
<td>assignment’s operands are arrays of chars</td>
</tr>
<tr>
<td>Array Reference</td>
<td>one array element is referred</td>
</tr>
<tr>
<td>Block Goto</td>
<td>the code execution goes to a new block</td>
</tr>
</tbody>
</table>
Learning source-code operation costs

Numbers of executions of the energy operations in one test case

\[
\begin{pmatrix}
  n_1^{(1)} & n_2^{(1)} & \ldots & n_l^{(1)} \\
  n_1^{(2)} & n_2^{(2)} & \ldots & n_l^{(2)} \\
  \vdots & \vdots & \ddots & \vdots \\
  n_1^{(m-1)} & n_2^{(m-1)} & \ldots & n_l^{(m-1)} \\
  n_1^{(m)} & n_2^{(m)} & \ldots & n_l^{(m)}
\end{pmatrix}
\]

Acquired from log file

Energy costs of the operations

\[
\begin{pmatrix}
  \text{cost}_1 \\
  \text{cost}_2 \\
  \vdots \\
  \text{cost}_l
\end{pmatrix}
\]

\[
\begin{pmatrix}
  e_1 \\
  e_2 \\
  \vdots \\
  e_{m-1} \\
  e_m
\end{pmatrix}
\]

Aiming to obtain

\[
\begin{pmatrix}
  \text{cost}_1 \\
  \text{cost}_2 \\
  \vdots \\
  \text{cost}_l
\end{pmatrix}
\times
\begin{pmatrix}
  n_1^{(1)} & n_2^{(1)} & \ldots & n_l^{(1)} \\
  n_1^{(2)} & n_2^{(2)} & \ldots & n_l^{(2)} \\
  \vdots & \vdots & \ddots & \vdots \\
  n_1^{(m-1)} & n_2^{(m-1)} & \ldots & n_l^{(m-1)} \\
  n_1^{(m)} & n_2^{(m)} & \ldots & n_l^{(m)}
\end{pmatrix}
\]

Measured
Learning the energy model

• Regression analysis, gradient descent to minimise an error function
• Four-fold cross validation
• Resulted in a model with Normalized Mean Absolute Error of about 15%

• 85% accuracy is a useful model for the software developer!
The energy model of source code operations

Top 10 ops account for 72.1% of energy usage.
Which code blocks use most energy?

A few blocks dominate energy usage. These are targets for energy optimisation.
Code optimisation

• Manual refactoring applied to hottest blocks
  – method inlining
  – code motion/code hoisting
  – loop unrolling
  – use of library functions

• The might in principle be performed by a compiler or refactoring tool, given the energ profile
Example optimisation

Program 5 The original code in CCWaves3D.update()

```java
int i, j;
for (i = 0; i < (gridSize.x+1); i++) {
    for (j = 0; j < (gridSize.y+1); j++) {
        CCVertex3D v = originalVertex( ccGridSize.ccg(i, j) );
        ...
        setVertex( ccGridSize.ccg(i, j), v );
    }
}
```

Program 6 Program 5 after Method Inline & Code Motion

```java
ccGridSize ccgridsize = new ccGridSize(0,0);//added
CCGrid3DAction ccgrid3d =
    (CCGrid3DAction) target.getGrid();  //added
CCVertex3D v = new CCVertex3D(0,0,0); //added
int i, j;
for (i = 0; i < (gridSize.x+1); i++) {
    for (j = 0; j < (gridSize.y+1); j++) {
        cgridsize.x=i; cgridsize.y=j;  //added
        v = ccgrid3d.originalVertex(ccgridsize);//changed
        ...
        ccgrid3d.setVertex(ccgridsize, v);//changed
    }
}
```
Optimisation. Example 1

Energy consumption of the code without and with the changes in Click & Move.

Overall saving: 6.4%
Optimisation. Example 2

Energy consumption of the code without and with the changes in Orbit.

Overall saving: 50.2%
Discussion

- A thorough energy analysis of a suite of code enabled insight into where most energy was consumed.
- This enabled source-code transformations to be focussed on the most effective areas.
Discussion (2)

• Complexity of the system stack does not need to be handled explicitly
  – rely on good machine learning methods
• Experiment limited to one class of applications (Cocos game engine apps).
  – No guarantee that the model would be good for other Android applications
  – However, the same techniques could be applied to other classes of application
• Only CPU power is measured, but could be extended to other parts of the platform
Future work

• Combine energy model with static resource analysis tools
  – Derive parametric energy consumption based on the source-code energy model
  – Soft bounds on energy consumption

• Apply similar model-learning methods in other complex architectures, e.g. distributed systems