May 18, 2015

ENTRA

Dynamic Energy Model of Application Source Code on Smart Devices

Xueliang Li

Roskilde University

UNIVERSITAS ROSKILDENSIS · IN TRANQUILLO MORS · IN FLUCTU VITA

University of BRISTOL

imdea instituto madrileño de estudios avanzados

XMOS®

COOPERATION

Motivation

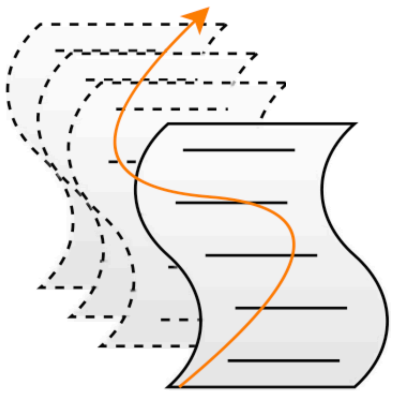
- Energy saving by a factor of 3 to 5 could be achieved by software optimizations alone. [C. Edwards , 2011]

- How to bridge the gap between high level source code and energy consumption?
  - Explore alternative approach to build a source code energy model.
  - Not based on a low level energy model

Approach

- Analyze a large set of test cases
  - Identify the set of basic energy-consuming operations
  - Obtain the execution paths of the cases
  - Label the path with actual measured energy cost
  - Employ the labeled data to train the energy model for the basic operations

Approach



energy consumption (e1)

Case1: op8 op3 op34 …… op1 op4 ……

e2

Case2: op2 op6 op1 …… op21 op5 ……

e i

Case i: op7 op2 op4 …… op1 op2 ……

• • • • • •

e m

Case m: op5 op7 op1 …… op9 op11 ……

$n_j^{(i)}$ = # executions of op j in case i

Model

- The model construction
  - based on data mining
  - correlate energy ops and energy costs from a large amount of data.

Model Construction

Numbers of executions of the energy operations in one test case

Energy costs of the operations

$$\begin{pmatrix} n_1^{(1)} & n_2^{(1)} & \dots & n_l^{(1)} \\ n_1^{(2)} & n_2^{(2)} & \dots & n_l^{(2)} \\ & \dots & \dots & \\ n_1^{(m-1)} & n_2^{(m-1)} & \dots & n_l^{(m-1)} \\ n_1^{(m)} & n_2^{(m)} & \dots & n_l^{(m)} \end{pmatrix} \times \begin{pmatrix} cost_1 \\ cost_2 \\ \dots \\ cost_l \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \\ \dots \\ e_{m-1} \\ e_m \end{pmatrix}$$

Aiming to obtain

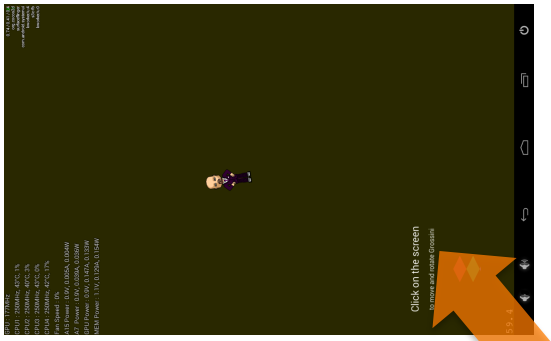
Acquired from log file

Measured

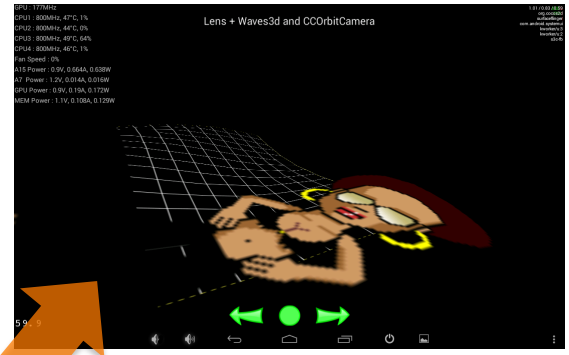
Measured energy consumption

Test Cases (Examples)

Click and move

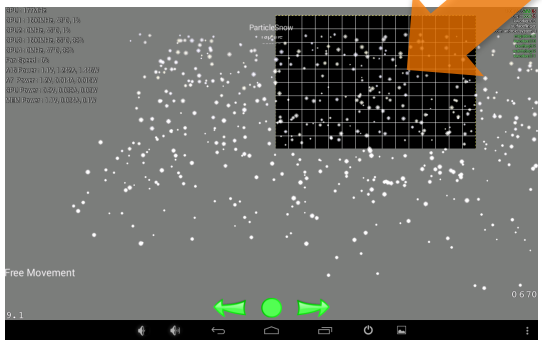


3D effect

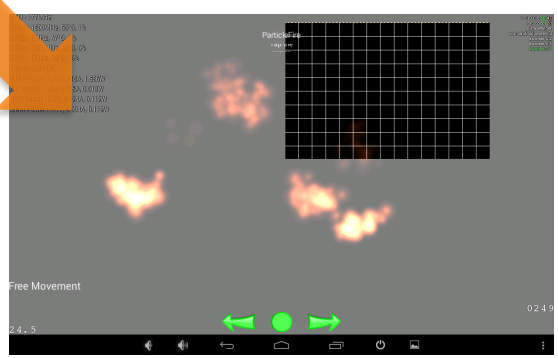


Input sequence 1: (tap, position1), (tap, position2).....
Input sequence 2: (tap, position1), (tap, position2).....
Input sequence 3: (tap, position1), (tap, position2).....
.....
Input sequence n: (tap, position1), (tap, position2).....

Snow



Fire



Experiment Setup

Target: Odroid-XU+E development board

- Two ARM quad-core CPUs, Cortex-A15 with 2.0 Ghz clock rate and Cortex-A7 with 1.5 Ghz.
- A built-in power monitor tool to measure the voltage and current of CPUs with a sampling frequency of 30 Hz

Source code: Cocos2d-Android

- A framework for building games, demos and other interactive applications.

Execution Path

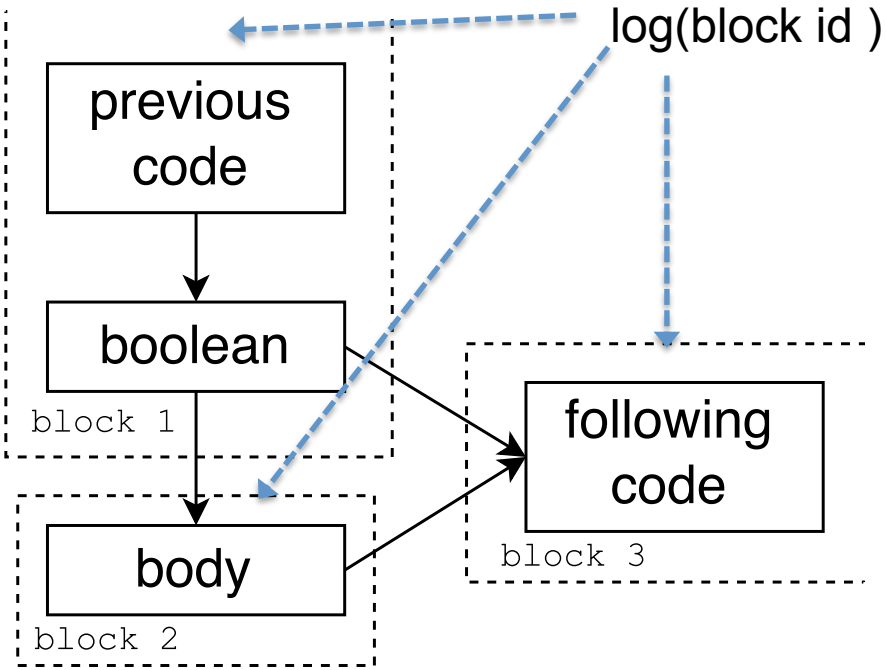
- We choose block as the basic unit of the path
  - Statement: impact significantly on energy cost and timing
  - Functions or classes: unstable execution components

Block Definition

- A block is a set of gathered statements.
  - In the block, each node has only one in-edge and one out-edge in the control flow graph,
  - But the start point of the block could have more than one in-edge,
  - The end point could have more than one out-edge.
- A block is a fixed execution unit. That means, always if one part of the block is processed, the rest certainly will be executed.

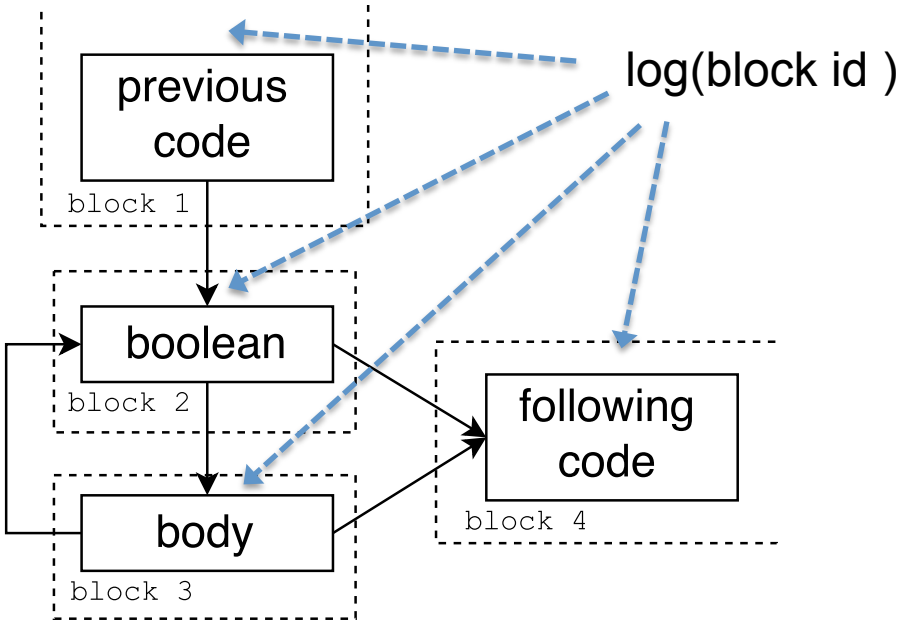
Block Division(If)

```
previous code;
if boolean then
  |  body;
end
following code;
```



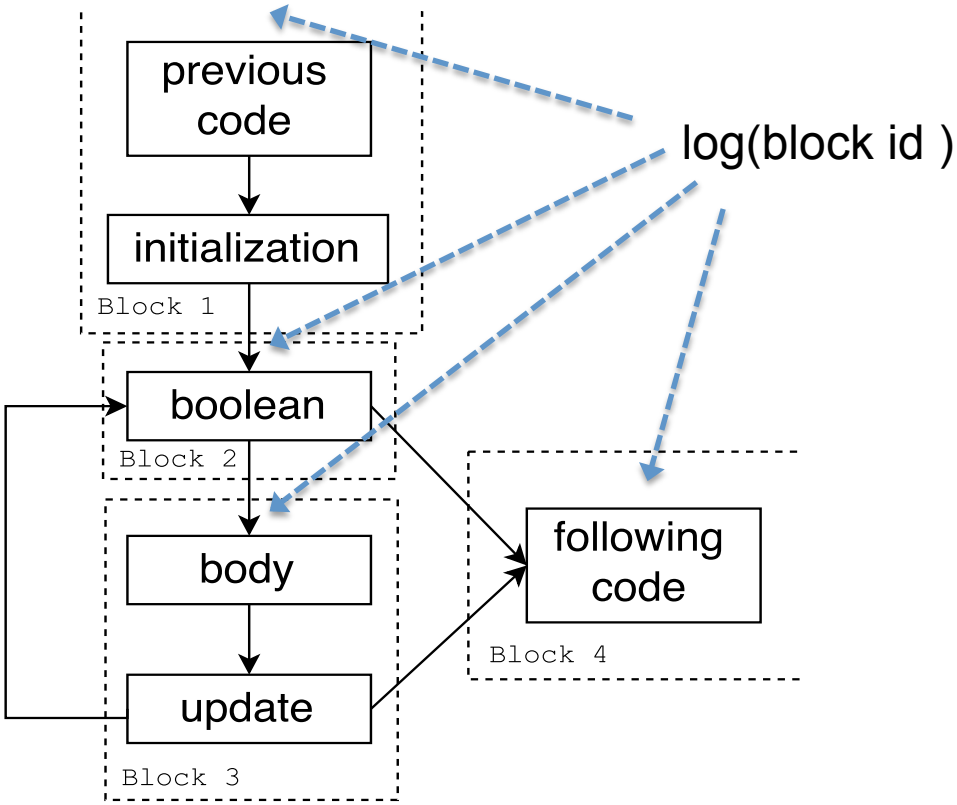
Block Division (While Loop)

```
previous code;
while boolean do
  |  body;
end
following code;
```



Block Division (For Loop)

previous code;
for *initialization; boolean; update* **do**
  body;
end
following code;



Operation Dictionary of Blocks

- We developed a parser to extract the energy ops from the source code. All of the ops are labeled with the ID of the block where it resides.

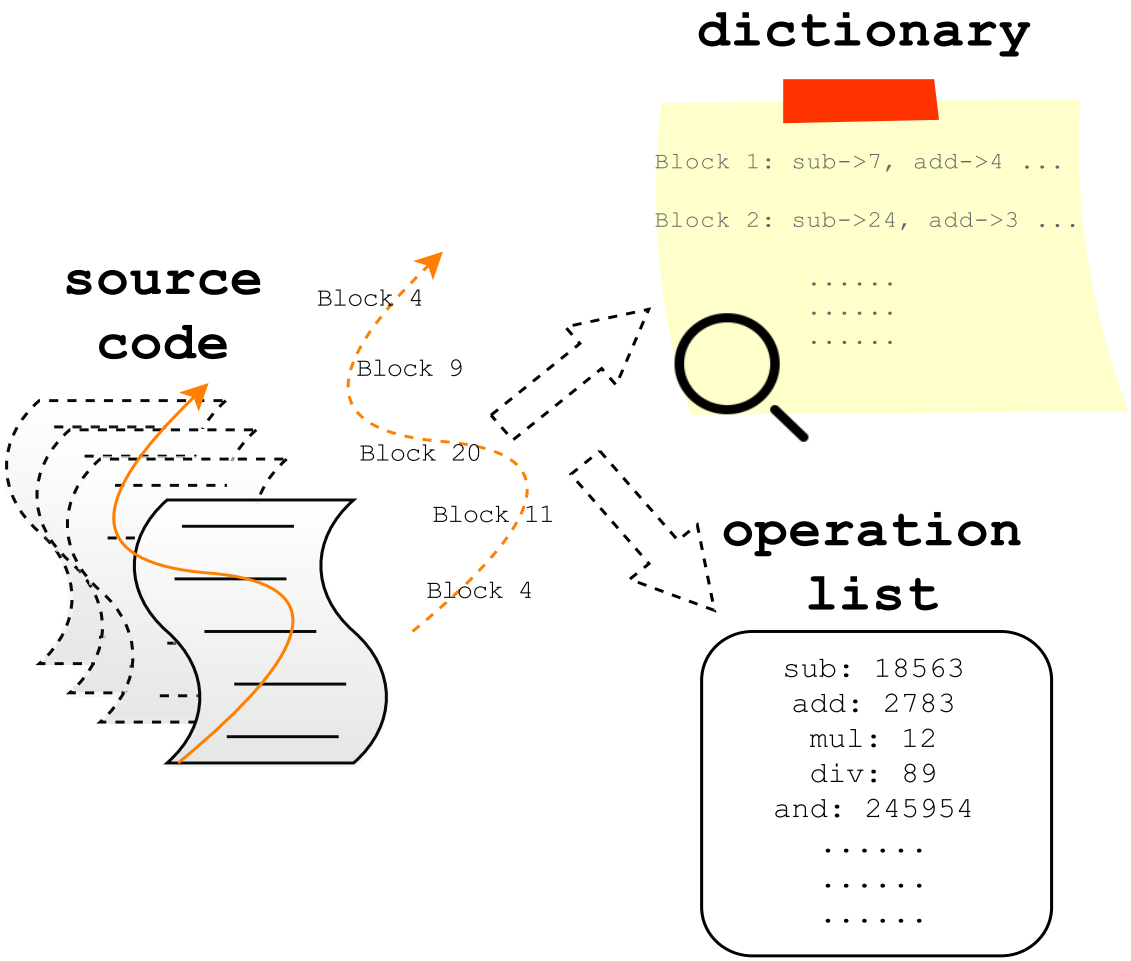
dictionary

```
Block 1: sub->7, add->4 ...

Block 2: sub->24, add->3 ...

      ......
      ......
      ......
```

Obtain Energy Operations on The Path



Model Construction

Numbers of executions of the energy operations in one test case

Energy costs of the operations

$$\begin{pmatrix} n_1^{(1)} & n_2^{(1)} & \dots & n_l^{(1)} \\ n_1^{(2)} & n_2^{(2)} & \dots & n_l^{(2)} \\ & \dots & \dots & \\ n_1^{(m-1)} & n_2^{(m-1)} & \dots & n_l^{(m-1)} \\ n_1^{(m)} & n_2^{(m)} & \dots & n_l^{(m)} \end{pmatrix} \times \begin{pmatrix} cost_1 \\ cost_2 \\ \dots \\ cost_l \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \\ \dots \\ e_{m-1} \\ e_m \end{pmatrix}$$

Aiming to obtain

Acquired from log file

Measured

Measured energy consumption

Error Function

$$\begin{pmatrix} n_1^{(1)} & n_2^{(1)} & \dots & n_l^{(1)} \\ n_1^{(2)} & n_2^{(2)} & \dots & n_l^{(2)} \\ & \dots & \dots & \\ n_1^{(m-1)} & n_2^{(m-1)} & \dots & n_l^{(m-1)} \\ n_1^{(m)} & n_2^{(m)} & \dots & n_l^{(m)} \end{pmatrix} \times \begin{pmatrix} cost_1 \\ cost_2 \\ \dots \\ cost_l \end{pmatrix} = \begin{pmatrix} e_1 \\ e_2 \\ \dots \\ e_{m-1} \\ e_m \end{pmatrix}$$

$$J(cost_1, cost_2, \dots cost_l) = \frac{1}{2m} \sum_{i=1}^{m} (\vec{n^{(i)}} \times \vec{cost} - e^{(i)})^2$$

Predicted Observed

Gradient Descent

To minimize: $$J(cost_1, cost_2, ...cost_l) = \frac{1}{2m}\sum_{i=1}^{m}(\vec{n^{(i)}} \times \vec{cost} - e^{(i)})^2$$

Repeat update until convergence:

$$cost_j := cost_j - \alpha \frac{\partial J(cost_1, ...cost_j, ...cost_l)}{\partial cost_j}$$

$$= cost_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(\vec{n^{(i)}} \times \vec{cost}) \cdot n_j^{(i)}$$

$$j = 1, 2, ...l$$

- The value α determines how large the step is in each iteration.

Correction part

The algorithm above may produce cost with negative elements, however as a matter of fact, the energy costs should be above zero.

Original part Correction part

$$J = \boxed{\frac{1}{2m}\sum_{i=1}^{m}(\vec{n^{(i)}} \times \vec{cost} - e^{(i)})^2} + \boxed{\lambda \frac{1}{l}\sum_{j=1}^{l}\rho^{-cost_j}}$$

$$\rho > 1$$

- The λ value balances the weights of correction part and that of the original part.
- The ρ value determines how aggressive the correction is

Experiment Progress

- 2700+ test cases
- Hard to obtain energy model at the "high level" and "middle level"

Next step:

- Go to lower level
- Consider the transition energy cost
- Use block as the model input

High level	Middle level	Lower
Arithmetic Ops	Addition, Subtraction Multiplication, Division Increment, Decrement	Information about the operands
Boolean Ops	And, Or, Not	
Comparison Ops	Greater, Less, Equal Greater or equal Less or equal	
Bitwise Ops	BitAnd, BitOr SignedBitShiftRight SignedBitShiftLeft	
Reference Ops	Array reference Field reference	
Function Ops	Argument passing Returning value	
Control Ops	Block goto Function Invocation	
Others	Declaration Type conversion	

Thank You！ Questions & Suggestions？