Power consumption profiling and optimization for embedded systems

Ekaterina Gorelkinia (e.gorelkinia@samsung.com)
Vasiliy Ulyanov (v.ulyanov@samsung.com)
Nelli Kim (nelly.kim@samsung.com)
MyungJoo Ham (myungjoo.ham@samsung.com)

Samsung R&D Institute Russia

October 25, 2013
Power consumption profiling and optimization

Introduction

**The main questions:**

1. How can we determine the *true* source of excessive power consumption?
2. What is the cause?
   - intensive CPU usage?
   - file or network operations?
   - graphics rendering?
   - ...

**The problem:** existing tools **cannot** give enough information for fast and easy power consumption bottlenecks location.

**The solution:** a tool that provides **detailed** problem description
   (the cause → application → function → source code line)
State of the art
Existing power consumption monitoring tools

Hardware (e.g. Power Monitor)

- System-level power consumption
- No per applications measurements
- No cause suggestion (CPU usage, IO, etc.)

Software (e.g. PowerTOP)

- Applications-level power consumption
- No per functions measurements
- No relative factors contribution at application level
Outline

1 Introduction

2 Power consumption monitoring tool (PoCoMon)

3 Testing

4 Usage example

5 Summary

6 Future work
Power consumption monitoring tool (PoCoMon)

Two steps method

**Step 1: Calibration**
- Power model
- Monitoring tool: power metrics
- HW power monitoring tool
- Coefficients

**Step 2: Estimation**
- Coefficients
- Monitoring tool: power metrics
- Power consumption estimation
Power consumption monitoring tool (PoCoMon)

Method basis

• *Dynamic binary instrumentation* approach:
  - kernel level profiling
  - applications/libraries functions instrumentation

• *Linear power model*:
  - the same for both system and applications power consumption profiling

• *Two steps* method:
  1. Calibration step:
     - needed for model coefficients estimation
     - performed once for a given device
  2. Power consumption estimation step:
     - can be performed after obtaining the coefficients at calibration step
     - no explicit need for a hardware monitoring device at this point
Power consumption monitoring tool (PoCoMon)

Method output

Power consumption for system/applications/functions

- Total power consumption estimation
- Individual factors contribution: CPU, IO, etc.
Method advantages

1. Extensible:
   - the linear power model allows to extend the set of factors depending on the developer’s needs

2. Easy to use:
   - calibration is performed only once for a given hardware
   - no external monitoring tool is required at estimation step

3. Detailed results:
   - problem location up to a "bad" function
   - cause description: which factor consumed the most
Dynamic binary instrumentation
SWAP: kprobe based tool

- kernel functions instrumentation
- application functions instrumentation
- runtime info collection
The power model can be represented as a linear combination of $N$ factors:

$$Q = \sum_{i=1}^{N} K_i r_i$$  \hfill (1)

Assuming:

- $Q$ is the consumed charge (in $mAh$)
- $K_i$ is the corresponding weight coefficient
- $r_i$ is the resource usage metric (e.g. CPU working time)

Depending on the needed precision the set of factors can be limited (e.g. considering only CPU usage and IO)
Calibration step

Overview

Example (Power model for CPU and IO usage: 4 factors)

\[ Q = K_{CPU} r_{CPU} + K_{IO} r_{IO} = (K_{cpu} t_{cpu} + K_{idle} t_{idle}) + (K_{read} b_{read} + K_{write} b_{write}) \] (2)

Assuming:

- \( K_{cpu} \) and \( K_{idle} \): [mA] - average electric current consumed by CPU (working/idle)
- \( K_{read} \) and \( K_{write} \): \( \frac{mAh}{byte} \) - average charge consumed per byte on IO operation

1. The goal: find the coefficients for the specified power model
2. The solution: use the least squares fitting
3. What we need:
   - Tests for each factor (e.g. for CPU, IO, etc.)
   - Resources usage metrics during test measurements (\( r_i \) values)
   - Measured power model function values (\( Q \))
Example (The four factors stress tests)

1. CPU stress test:

```c
while (1) {
    sqrt(123.45);
}
```

2. Idle test:

   *no explicit workload*

3. File read test:

   ```
   dd if=/dev/mmcblk0p7 of=/dev/null bs=32M count=32
   ```

4. File write test:

   ```
   dd if=/dev/zero of=/opt/usr/test.out bs=32M count=32 conv=fdatasync
   ```
### Calibration step

**Resources usage metrics**

**Table**: Resource usage metrics

<table>
<thead>
<tr>
<th>#</th>
<th>Metric</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPU usage time</td>
<td>$t_{cpu}$</td>
<td>CPU working time for <strong>all</strong> processes <em>(except Idle)</em></td>
</tr>
<tr>
<td>2</td>
<td>Idle time</td>
<td>$t_{idle}$</td>
<td>CPU time spent on the Idle process</td>
</tr>
<tr>
<td>3</td>
<td>Read amount</td>
<td>$b_{read}$</td>
<td>number of bytes read (taken from function’s arguments)</td>
</tr>
<tr>
<td>4</td>
<td>Write amount</td>
<td>$b_{write}$</td>
<td>number of bytes written (taken from function’s arguments)</td>
</tr>
</tbody>
</table>
Calibration step
Power model function values

Consumed charge can be obtained by sampling instant electric current values and then integrating them by time:

$$Q = \int_{t_1}^{t_n} I(t)dt \approx \frac{1}{2} \sum_{i=1}^{n-1} (t_{i+1} - t_i)(I_{i+1} + I_i)$$

(3)

Possible discrete $I_i$ values source:

- External *hardware* power monitoring tool
- Linux kernel *power_supply* subsystem:

Example

```c
union power_supply_propval propval;
struct power_supply *psu = power_supply_get_by_name("...");
psu->get_property(psu, POWER_SUPPLY_PROP_CURRENT_NOW, &propval);
```
Calibration step

Least squares fitting: \( Q = \sum_{i=1}^{N} K_i r_i \)

- **Observations:**

\[
X = \begin{pmatrix}
t_{cpu_1} & t_{idle_1} & b_{read_1} & b_{write_1} \\
t_{cpu_2} & t_{idle_2} & b_{read_2} & b_{write_2} \\
\vdots & \vdots & \vdots & \vdots \\
t_{cpu_n} & t_{idle_n} & b_{read_n} & b_{write_n}
\end{pmatrix}
\]

(4)

- **Measurements:**

\[
y = (Q_1 \ Q_2 \ \cdots \ Q_n)^T
\]

(5)

- **Coefficients:**

\[
K = (K_{cpu} \ K_{idle} \ K_{read} \ K_{write})^T
\]

(6)

- **Least squares fitting:**

\[
K = (X^T X)^{-1} X^T y
\]

(7)
Estimation step

Overview

Example (Power model for CPU and IO usage: 4 factors)

\[ Q = K_{\text{cpu}} t_{\text{cpu}} + K_{\text{idle}} t_{\text{idle}} + K_{\text{read}} b_{\text{read}} + K_{\text{write}} b_{\text{write}} \]  (8)

1. **The goal**: given the power model with parameters find the consumed charge for every instrumented application and function

2. **The solution**: bind the resources usage metrics to application functions

3. **What we need**:  
   - kernel instrumentation for power metrics calculation  
   - application functions profiling  
   - power model with known parameters
Estimation step

Resources usage metrics

Table: Resource usage metrics

<table>
<thead>
<tr>
<th>#</th>
<th>Metric</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CPU usage time</td>
<td>( t_{cpu} )</td>
<td>CPU working time for each process</td>
</tr>
<tr>
<td></td>
<td>Read amount</td>
<td>( b_{read} )</td>
<td>number of bytes read (taken from function's arguments)</td>
</tr>
<tr>
<td></td>
<td>Write amount</td>
<td>( b_{write} )</td>
<td>number of bytes written (taken from function's arguments)</td>
</tr>
<tr>
<td></td>
<td>Userspace events</td>
<td>( t_{entry}/t_{return} )</td>
<td>functions entry/return time</td>
</tr>
</tbody>
</table>
Estimation step
Applications level: CPU usage time

Example
- **Idle process:** $t_{cpu} = (t_1 - t_0) + (t_6 - t_5)$
- **Process 1:** $t_{cpu} = (t_2 - t_1) + (t_4 - t_3)$

`switch_to` scheduler function instrumentation:
- **entry time:** process inactive
- **return time:** process active
### Estimation step

Applications level: IO amount

#### Example (Collected trace: system read/write operations)

<table>
<thead>
<tr>
<th>Time</th>
<th>Function</th>
<th>Type</th>
<th>Process</th>
<th>Args/retval</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:02.478034</td>
<td>sys_read</td>
<td>ENTRY</td>
<td>bzip2</td>
<td>'/opt/usr/test.inp', 0xb6ff0000, 4096, 'mmcblk0'</td>
</tr>
<tr>
<td>00:02.478058</td>
<td>sys_read</td>
<td>RETURN</td>
<td>bzip2</td>
<td>4096</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00:03.428031</td>
<td>sys_write</td>
<td>ENTRY</td>
<td>bzip2</td>
<td>'/opt/usr/test.out', 0xb6f6f000, 4096, 'mmcblk0'</td>
</tr>
<tr>
<td>00:03.428131</td>
<td>sys_write</td>
<td>RETURN</td>
<td>bzip2</td>
<td>4096</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Estimation step
Functions level: CPU usage time

Application functions instrumentation:
- entry: function call time
- return: function return time

Example
- Process 1: $t_{cpu} = (t_0 - t_{entry}) + (t_2 - t_1) + (t_{return} - t_3)$
### Example (Collected trace: per-function read/write operations)

<table>
<thead>
<tr>
<th>Time</th>
<th>Function</th>
<th>Type</th>
<th>Process</th>
<th>Args/retval</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>compressStream</td>
<td>ENTRY</td>
<td>bzip2</td>
<td></td>
</tr>
<tr>
<td>00:02.475</td>
<td>sys_read</td>
<td>ENTRY</td>
<td>bzip2</td>
<td>'/opt/usr/test.inp', 0xb6ff0000, 4096, 'mmcblk0'</td>
</tr>
<tr>
<td>00:02.478</td>
<td>sys_read</td>
<td>RETURN</td>
<td>bzip2</td>
<td>4096</td>
</tr>
<tr>
<td>00:02.478</td>
<td>BZ2_bzWriteClose64</td>
<td>ENTRY</td>
<td>bzip2</td>
<td>'/opt/usr/test.out', 0xb6fef000, 4096, 'mmcblk0'</td>
</tr>
<tr>
<td>00:03.428</td>
<td>sys_write</td>
<td>ENTRY</td>
<td>bzip2</td>
<td>4096</td>
</tr>
<tr>
<td>00:03.428</td>
<td>sys_write</td>
<td>RETURN</td>
<td>bzip2</td>
<td>0</td>
</tr>
<tr>
<td>00:03.428</td>
<td>BZ2_bzWriteClose64</td>
<td>RETURN</td>
<td>bzip2</td>
<td>0</td>
</tr>
<tr>
<td>00:04.106</td>
<td>compressStream</td>
<td>RETURN</td>
<td>bzip2</td>
<td>0</td>
</tr>
</tbody>
</table>
System consumption: *bzip2* use case

System consumption is composed of:

- *bzip2* (main activity)
- *mmcqrd* (background worker)
- *idle* consumption
- *other processes*
**Test cases**

bzip2 compress/decompress

<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>mAh</th>
<th>Ncalls</th>
<th>CPU, %</th>
<th>Read, %</th>
<th>Write, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mainSort</td>
<td>0.82</td>
<td>117</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>BZ2_compressBlock</td>
<td>0.81</td>
<td>117</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>handle_compress</td>
<td>0.09</td>
<td>41945</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>BZ2_bzWrite</td>
<td>0.03</td>
<td>20972</td>
<td>60.65</td>
<td>0.00</td>
<td>39.35</td>
</tr>
<tr>
<td>5</td>
<td>compressStream</td>
<td>0.02</td>
<td>1</td>
<td>66.90</td>
<td>33.10</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>add_pair_to_block</td>
<td>0.01</td>
<td>407361</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**System power consumption**

est: 2.16 | real: 2.28 | err: 5%

<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>mAh</th>
<th>Ncalls</th>
<th>CPU, %</th>
<th>Read, %</th>
<th>Write, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BZ2_decompress</td>
<td>0.35</td>
<td>21183</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>BZ2_bzDecompress</td>
<td>0.31</td>
<td>42037</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>uncompressStream</td>
<td>0.04</td>
<td>1</td>
<td>75.55</td>
<td>0.00</td>
<td>24.45</td>
</tr>
<tr>
<td>4</td>
<td>BZ2_bzRead</td>
<td>0.02</td>
<td>20972</td>
<td>74.17</td>
<td>25.83</td>
<td>0.00</td>
</tr>
</tbody>
</table>

**System power consumption**

est: 0.88 | real: 1.05 | err: 15%

System-level power consumption estimation error: **5-15%**
Outline

1. Introduction
2. Power consumption monitoring tool (PoCoMon)
3. Testing
4. Usage example
5. Summary
6. Future work
Example: incorrect function usage

**objgetattr** test case

Example (**objgetattr** incorrect usage)

```c
obj_lock(obj); // mutual exclusive access
while (cnt > 0) {
    ...
    /*
    * obj_getattr: some framework function.
    * The implementation is hidden from the
    * end developer.
    * Internally it may invoke IO operations.
    * */
    do_work1(obj_getattr(obj));
    ...
    cnt --;
}
obj_unlock(obj);
```
Example: incorrect function usage

Solution

Example (move `obj_getattr` out from the loop)

```c
obj_lock(obj); // mutual exclusive access
attr = obj_getattr(obj);
while (cnt > 0) {
    ...
    do_work1(attr);
    ...
    cnt--;
}
obj_unlock(obj);
```
Example: incorrect function usage

Comparison

Table: `obj_getattr` test case comparison

<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>mAh</th>
<th>Ncalls</th>
<th>CPU, %</th>
<th>Read, %</th>
<th>Write, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>obj_getattr</td>
<td>3.17</td>
<td>50000</td>
<td>12.77</td>
<td>87.23</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>main</td>
<td>0.02</td>
<td>1</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>do_work1</td>
<td>0.01</td>
<td>50000</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Original consumption: **3.19 mAh**

<table>
<thead>
<tr>
<th>#</th>
<th>Function</th>
<th>mAh</th>
<th>Ncalls</th>
<th>CPU, %</th>
<th>Read, %</th>
<th>Write, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>main</td>
<td>0.02</td>
<td>1</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>do_work1</td>
<td>0.01</td>
<td>50000</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>obj_getattr</td>
<td>0.00</td>
<td>1</td>
<td>86.51</td>
<td>13.49</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Optimized consumption: **0.03 mAh**
1. Introduction
2. Power consumption monitoring tool (PoCoMon)
3. Testing
4. Usage example
5. Summary
6. Future work
PoCoMon: A software Power Consumption Monitoring tool designed to perform analysis at applications level.

- dynamic binary instrumentation approach
- linear power model
- two steps method:
  1. calibration
  2. estimation
- per functions measurements
- relative factors contribution in total power consumption
Outline

1. Introduction
2. Power consumption monitoring tool (PoCoMon)
3. Testing
4. Usage example
5. Summary
6. Future work
Future work

1. CPU frequency scaling handling; multicore support
2. Additional devices support
   - GPU
   - WiFi
   - Bluetooth
   - Sensors
   - ...
3. Use recursive adaptive filters at estimation step for better parameters fitting with a particular use case (runtime coefficients adjusting)
4. General error estimation, likelihood-ratio test
5. User friendly interface
6. Any other ideas/suggestions are welcome!
Thank you for your attention. Any questions?