Power consumption profiling and optimization

for embedded systems

Ekaterina Gorelkina (e.gorelkina@samsung.com)
Vasiliy Ulyanov (v.ulyanov@samsung.com)
Nelli Kim (nelli.kim@samsung.com)
MyungJoo Ham (myungjoo.ham@samsung.com)

Samsung R&D Institute Russia

October 25, 2013

Outline

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)

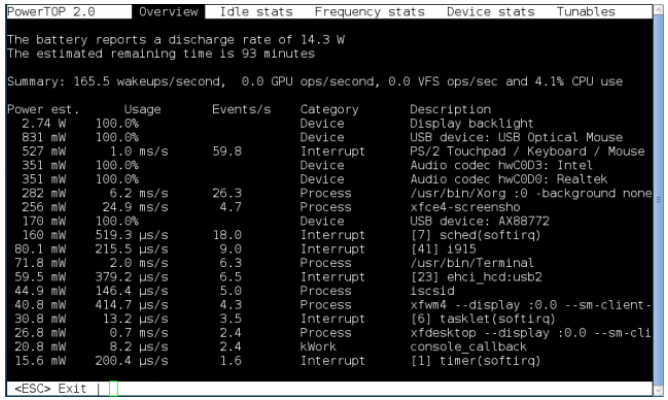
State of the art

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)

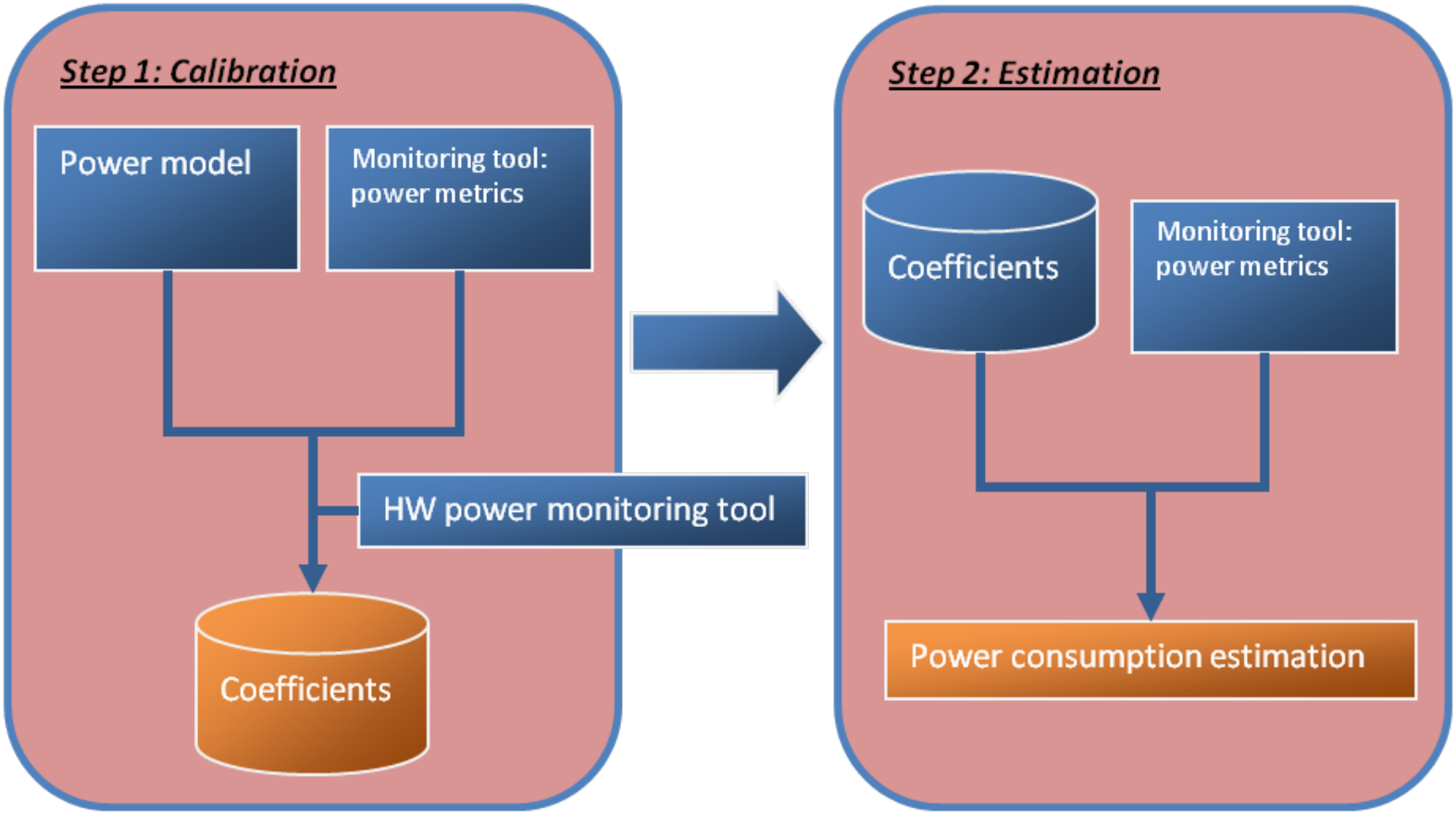


- *applications-level power consumption*
- no per functions measuremetns
- no relative factors contribution at application level

Outline

Power consumption monitoring tool (PoCoMon)

Two steps method

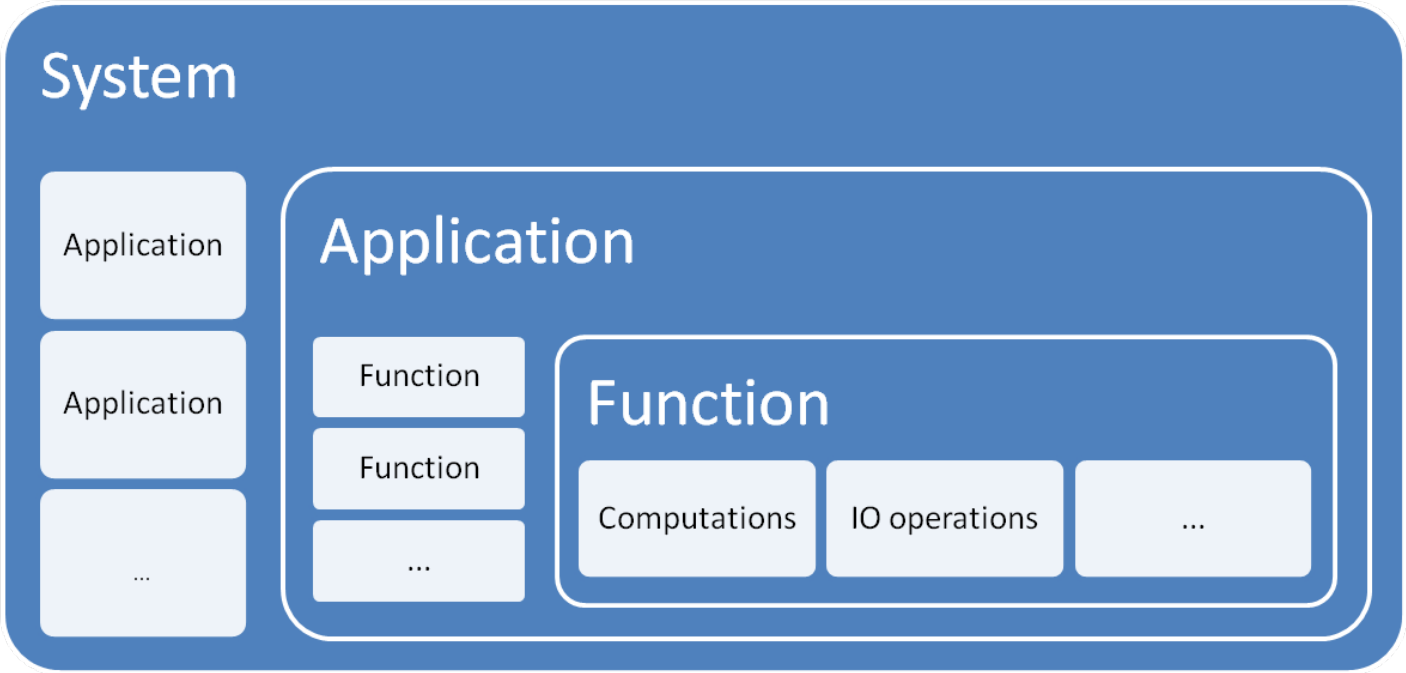


Power consumption monitoring tool (PoCoMon)

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 for *system/applications/functions*



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

Power consumption monitoring tool (PoCoMon)

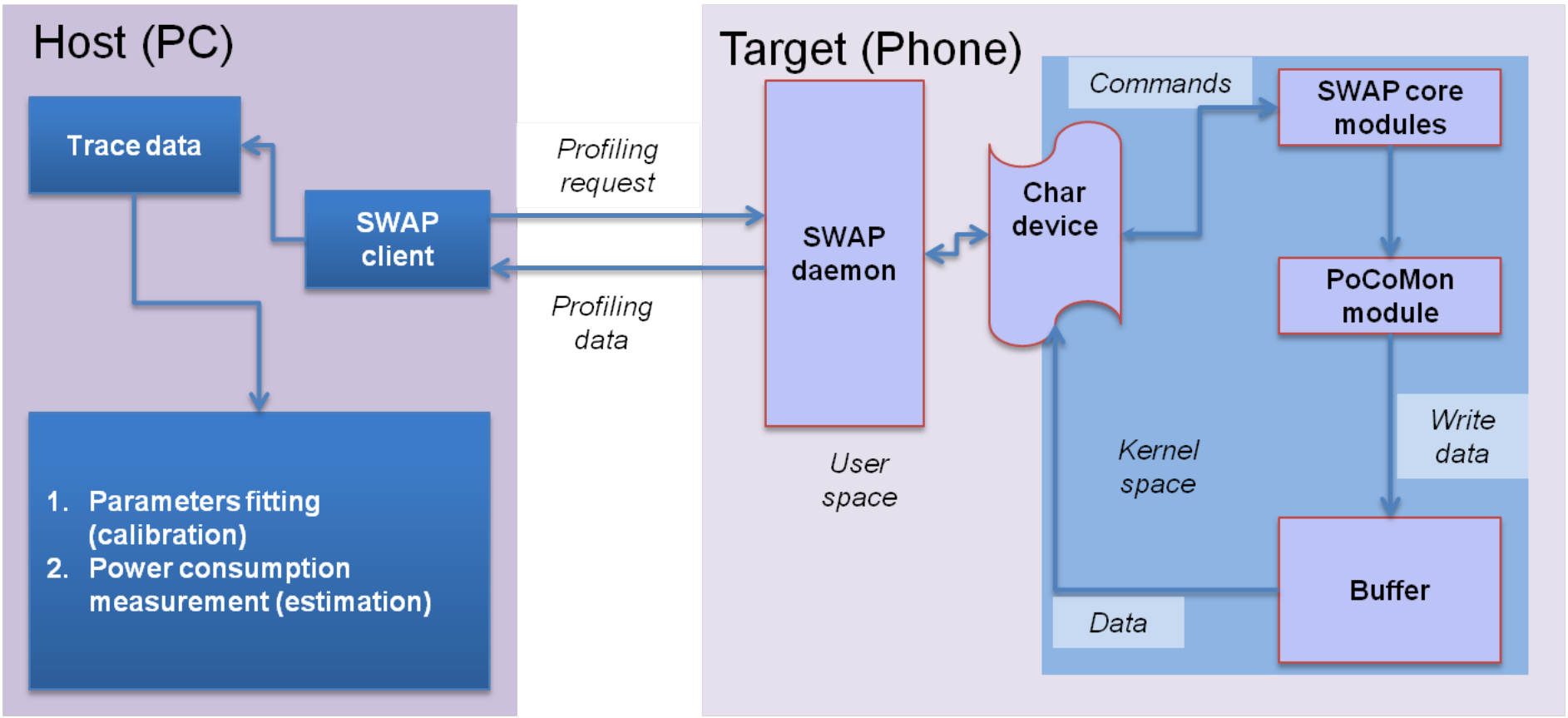
Method advantages

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



Power model

The power model can be represented as a linear combination of N factors:

$$Q = \sum_{i=1}^{N} K_i r_i \tag{1}$$

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)

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

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)

Calibration step

Running stress tests

Example (The four factors stress tests)

1. CPU stress test:

```
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

Table : Resource usage metrics

#	Metric		Function	Description
1	CPU usage time	t_{cpu}	__switch_to	CPU working time for **all** processes (**except Idle**)
2	Idle time	t_{idle}	__switch_to	CPU time spent on the Idle process
3	Read ammount	b_{read}	sys_read	number of bytes read (taken from function's arguments)
4	Write ammount	b_{write}	sys_write	number of bytes written (taken from function's arguments)

Calibration step

Power model function values

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

$$Q = \int_{t_1}^{t_n} I(t)dt \approx \frac{1}{2}\sum_{i=1}^{n-1}(t_{i+1} - t_i)(I_{i+1} + I_i) \tag{3}$$

Possible discrete I_i values source:

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

Example

```
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*:

$$X = \begin{pmatrix} t_{cpu_1} & t_{idle_1} & b_{read_1} & b_{write_1} \\ t_{cpu_2} & t_{idle_2} & b_{read_2} & b_{write_2} \\ \vdots & \vdots & \vdots & \vdots \\ t_{cpu_n} & t_{idle_n} & b_{read_n} & b_{write_n} \end{pmatrix} \tag{4}$$

- *Measurements*:

$$y = \begin{pmatrix} Q_1 & Q_2 & \cdots & Q_n \end{pmatrix}^T \tag{5}$$

- *Coefficients*:

$$K = \begin{pmatrix} K_{cpu} & K_{idle} & K_{read} & K_{write} \end{pmatrix}^T \tag{6}$$

- *Least squares fitting*:

$$K = (X^T X)^{-1} X^T y \tag{7}$$

Estimation step

Overview

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

$$Q = K_{cpu} t_{cpu} + K_{idle} t_{idle} + K_{read} b_{read} + K_{write} b_{write} \quad (8)$$

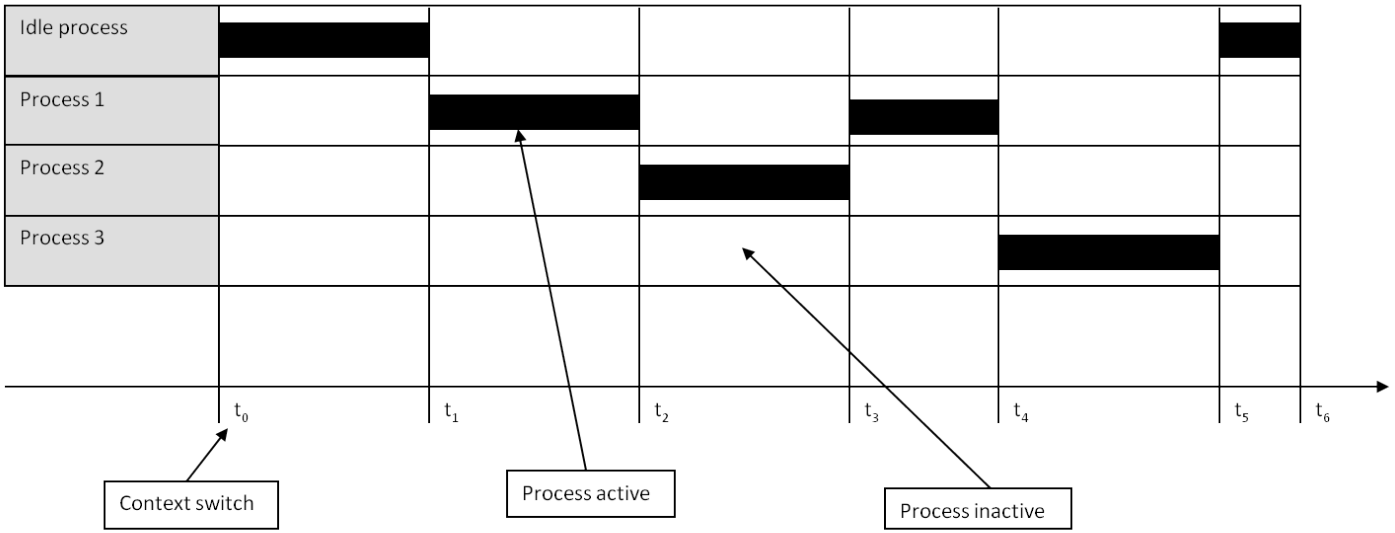
1. *The goal*: given the power model with parameters find the consumed charge for every instrumented application and function
2. *The solution*: bind the resources usage metrics to application functions
3. *What we need*:
   - kernel instrumentation for power metrics calculation
   - application functions profiling
   - power model with known parameters

Table : Resource usage metrics

#	Metric		Function	Description
1	CPU usage time	t_{cpu}	__switch_to	CPU working time for **each** process
2	Read ammount	b_{read}	sys_read	number of bytes read (taken from function's arguments)
3	Write ammount	b_{write}	sys_write	number of bytes written (taken from function's arguments)
4	Userspace events	t_{entry}/t_{return}	all application functions	functions entry/return time

Estimation step

Applications level: CPU usage time



__switch_to scheduler function instrumentation:

- *entry time*: process inactive
- *return time*: process active

Example

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

Estimation step

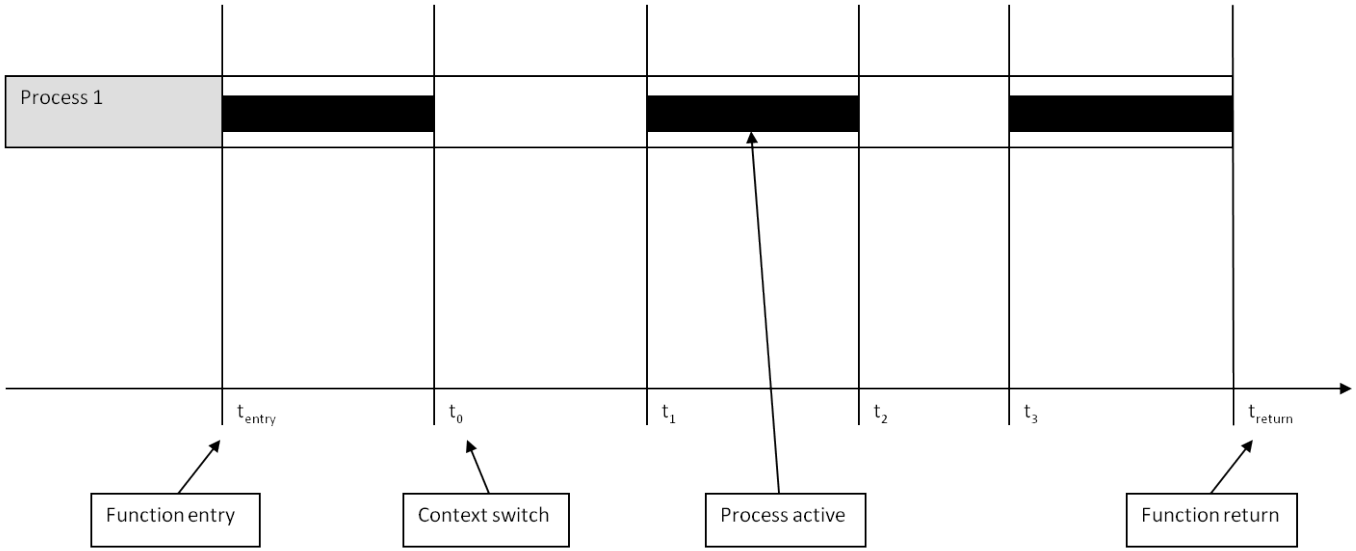
Applications level: IO ammount

Example (Collected trace: system read/write operations)

Time	Function	Type	Process	Args/retval
...				
00:02.478034	sys_read	ENTRY	bzip2	'/opt/usr/test.inp', 0xb6ff0000, 4096, 'mmcblk0'
00:02.478058	sys_read	RETURN	bzip2	**4096**
...				
00:03.428031	sys_write	ENTRY	bzip2	'/opt/usr/test.out', 0xb6fef000, 4096, 'mmcblk0'
00:03.428131	sys_write	RETURN	bzip2	**4096**
...				

Estimation step

Functions level: CPU usage time



Application functions instrumentation:

- *entry*: function call time
- *return*: function return time

Example

- *Process 1:* $t_{cpu} = (t_0 - t_{entry}) + (t_2 - t_1) + (t_{return} - t_3)$

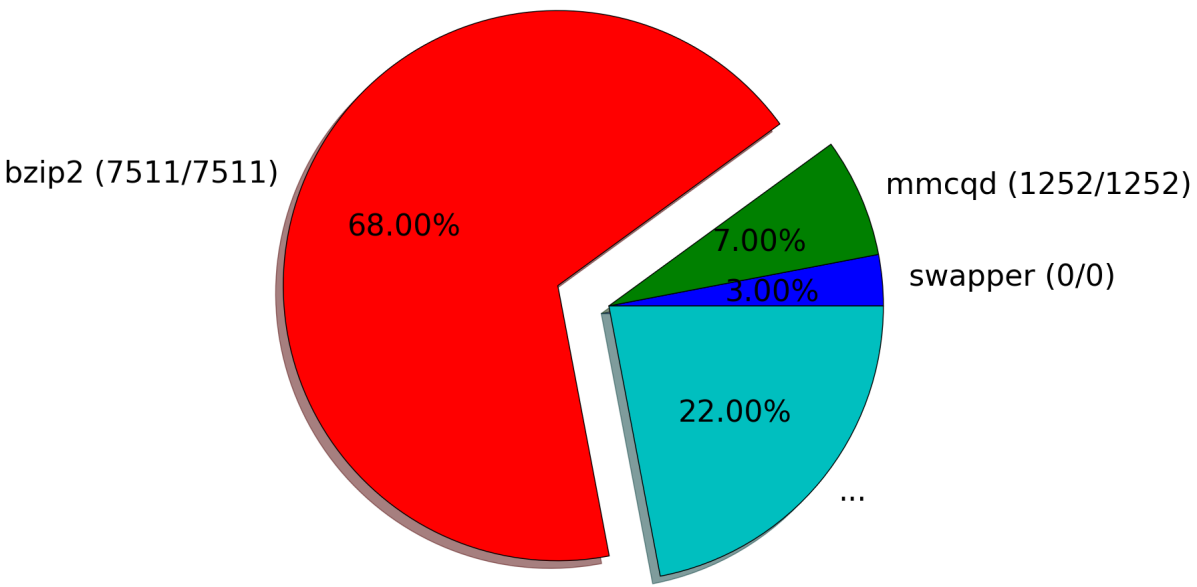
Estimation step

Functions level: IO ammount

Example (Collected trace: per-function read/write operations)

Time	Function	Type	Process	Args/retval
...				
⊢00:02.475323	compressStream	ENTRY	bzip2	...
⊢00:02.478034	sys_read	ENTRY	bzip2	'/opt/usr/test.inp', 0xb6ff0000, 4096, 'mmcblk0'
⊢00:02.478058	sys_read	RETURN	bzip2	**4096**
⊢00:13.385030	BZ2_bzWriteClose64	ENTRY	bzip2	...
⊢00:03.428031	sys_write	ENTRY	bzip2	'/opt/usr/test.out', 0xb6fef000, 4096, 'mmcblk0'
⊢00:03.428131	sys_write	RETURN	bzip2	**4096**
⊢00:14.105588	BZ2_bzWriteClose64	RETURN	bzip2	0
⊢00:14.105635	compressStream	RETURN	bzip2	0
...				

System consumption: *bzip2* use case



System consumption is composed of:

- *bzip2* (main activity)
- *mmcqd* (background worker)
- *idle* consumption
- *other processes*

Outline

Test cases

bzip2 compess/decompress

#	Function	mAh	Ncalls	CPU, %	Read, %	Write, %
#1: **bzip2 -c random.data** > **random.data.bz2** (1.79 mAh/82.98%)						
1	mainSort	0.82	117	100.00	0.00	0.00
2	BZ2_compressBlock	0.81	117	100.00	0.00	0.00
3	handle_compress	0.09	41945	100.00	0.00	0.00
4	BZ2_bzWrite	0.03	20972	60.65	0.00	39.35
5	compressStream	0.02	1	66.90	33.10	0.00
6	add_pair_to_block	0.01	407361	100.00	0.00	0.00
System power consumption				**est: 2.16**	**real: 2.28**	**err: 5%**
#2: **bzip2 -dc random.data.bz2** > **test.out** (0.72 mAh/82.06%)						
1	BZ2_decompress	0.35	21183	100.00	0.00	0.00
2	BZ2_bzDecompress	0.31	42037	100.00	0.00	0.00
3	uncompressStream	0.04	1	75.55	0.00	24.45
4	BZ2_bzRead	0.02	20972	74.17	25.83	0.00
System power consumption				**est: 0.88**	**real: 1.05**	**err: 15%**

System-level power consuption estimation error: **5-15%**

Outline

Example: incorrect function usage

obj_getattr test case

Example (*obj_getattr* incorrect usage)

```
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)

```
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

#	Function	mAh	Ncalls	CPU, %	Read, %	Write, %
1	obj_getattr	3.17	50000	12.77	87.23	0.00
2	main	0.02	1	100.00	0.00	0.00
3	do_work1	0.01	50000	100.00	0.00	0.00
Original consumption: **3.19 mAh**						
1	main	0.02	1	100.00	0.00	0.00
2	do_work1	0.01	50000	100.00	0.00	0.00
3	obj_getattr	0.00	1	86.51	13.49	0.00
Optimized consumption: **0.03 mAh**						

Outline

Summary

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

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!

Q&A

Thank you for your attention.
Any questions?