Clustering and Symbolic Regression for Power Consumption Estimation on Smartphone Hardware Subsystems

Ekarat Rattagan, Ying-Dar Lin, *Fellow, IEEE,* Yuan-Cheng Lai, Edward T.-H. Chu, and Kate Ching-Ju Lin

Abstract—The subsystem in a smartphone means its hardware components, such as the CPU, GPU, and screen. Accurately estimating subsystem power consumption of commercial smartphones is necessary for applicable to wide research areas. Current subsystem power estimation techniques are mostly based on power models, resulting in considerable errors for various types of power consumption behaviors. These include (1) asynchrony between the measured power consumption and the corresponding workload statistics, and (2) nonlinearity concerning CPU idle states, pixels colors of AMOLED screen, and GPU workload statistics. In this study we propose a novel utilization-based, subsystem power estimation method for a smartphone, namely Clustering and Symbolic Regression (CSR) that takes these power consumption behaviors into account so as to increase power estimation accuracy. To address asynchrony, we cluster the subsystem workload statistics into synchronous and asynchronous groups by employing affinity propagation clustering. To address nonlinearity, we employ symbolic regression for fitting measured power consumptions with respect to subsystem workload statistics. We compare our approach with various power estimation methods, Linear Regression Model (LM), Genetic Programming (GP), and Support Vector Regression (SVR). The results show Mean Absolute Percentage Error (MAPE) reduction between 23.61% and 42.55% on the estimated power consumption of a simple (Nexus S) and complex (Galaxy S4) smartphone subsystems

Index Terms—Smartphone subsystems, evolutionary computation, clustering methods, power consumption modeling and estimation

1 Introduction

Fast battery draining is the most critical issue for today smartphones, and smartphone applications play an important role in the cause of this major issue [1]. Without clearly understanding the power consumption behaviors of smartphone applications, developers may inadvertently cause their applications to drain excessive power. It is thus necessary for application developers to be aware of the power usage of the applications they develop. Developers can then monitor the power consumption of applications from a power profile which gives the power consumption information of smartphone subsystems as provided by manufacturers such as an Android power profile [2]. However, the power profile provided causes significant errors in old smartphones. Dong and Zhong [3] determined the causes of inaccuracies in generated power profiles and suggested that the power profile of each smartphone should frequently be reconstructed to reduce its inaccuracy.

In general, a power profile generated by smartphone vendors contains a list of the correlation between subsystem workload statistics and the associated measured power consumption. However, reconstructing a power profile on a commercial smartphone is labor-intensive, especially for the power measurement task, as the manufacturer does not provide any schematics for power measurement or a way of measuring the consumption of each. Hence, to obtain the power consumption of the target subsystems, most existing works employ a subtractive method, which works by subtracting the power consumption of the other subsystems (obtained from the generated subsystem power models) from the total system power consumption (obtained from an external power meter).

Recently several studies have proposed subsystem power estimation modeling for commercial smartphones. These studies can be classified into two categories: utilization-based methods and instrumentation-based methods. Utilization-based methods refer to the profilers that collect statistical data at a regular interval, whereas instrumentation-based methods collect the required information when a specific event occurs. In utilization-based methods, Shy et al. [4] and Zhang et al. [5] used linear regressions to build a power model of all major subsystems. Kjrgaard and Blunck [6] applied a genetic algorithm, whereas Ma et al. [7] used support vector regression to build a power model of the subsystem, GPU, which nonlinearly consumes power. On the other hand, in instrumentation-based methods, Pathak et al. [8] stated that the tail energy, a type of asynchronous power, consumed significant power, and proposed system call-tracing to detect tail energy on some subsystems such as GPS, SD-card, Wi-Fi, and 3G.

- *E. Rattagan is with the Faculty of Information Science and Technology, Mahanakorn University of Technology, Thailand. E-mail: rekara40@mut.ac.th*
- *Y. D. Lin and K. C.-J. Lin are with Department of Computer Science, National Chiao Tung University, Taiwan. E-mail:ydlin, katelin@cs.nctu.edu.tw*
- *Y. C. Lai is with Department of Information Management, National Taiwan University of Science and Technology, Taiwan. E-mail: laiyc@cs.ntust.edu.tw*
- *E. T.-H. Chu is with Department of Computer Science and Information Engineering, National Yunlin University of Science and Technology, Taiwan. E-mail: edwardchu@yuntech.edu.tw*

Manuscript received ; revised .

Asynchronous power (ASP) is defined as the consumed power which is uncorrelated with the corresponding subsystem workload statistics. Cao et al. [9] instrumented the browser in order to use WEB page load activities and resource information for modeling and predicting the energy consumption of mobile Web page.

Although existing power modeling techniques can work in general cases, they still have some limitations in handling the power consumption behavior of complex subsystems, leading to inaccurate power estimates. For example, the linear regression techniques may generate good results for a CPU where only two parameters (utilization and frequency) are considered. However, they cause significant errors when additional CPU parameters (idle time and entries) are considered [10]. These errors are the result of the presence of nonlinear power consumption behaviors of CPU idle time and entries. Nonlinear power (NLP) is defined as the consumed power which is not linearly proportional to the workload statistics. Moreover, ASP occurs in some complex subsystems, such as GPU, which are too complex to be detected by instrumentation-based approaches. Furthermore, most of the existing works rely on those power models whose forms (mathematical equations) are automatically discovered by mathematical modeling methods from machine learning or artificial intelligence techniques, e.g., linear and nonlinear regressions, genetic programming, support vector regression, etc. (named as a machine-defined model form). Compared with a model whose forms are manually defined by a human (named as a human-defined model form), the traditional machine-defined model form approach is only suitable for expressing the simple power consumption behaviors. However, they are not applicable to the complex power consumption behavior of modern smartphones since they are associated with many parameters, e.g., a GPU is associated with about 19 parameters. Generally speaking, the human-defined model form can manually provide a more accurate model, while traditional machine-defined model form can automatically generate a less accurate model.

In this paper, we propose a novel utilization-based power estimation method, called Clustering and Symbolic Regression (CSR), to deal with ASP and NLP to improve the estimation accuracy. To determine ASP, a clustering method is introduced to first classify the data into samples correlated with synchronous or asynchronous. In this work, the Affinity Propagation (AP) clustering algorithm [11] is applied since AP does not need to specify the number of clusters in advance. This property of AP is suitable for discovering ASP whose number of occurrences is difficult to predict. CSR aims to detect ASP on various subsystems, especially for a complex subsystem such as GPU, because it is impractical to apply instrumentation-based approaches since GPU source code is closed. Next, the obtained data sample, excluding ASP, is passed through the symbolic regression (SR) method of Eureqa software [12] in order to build a power model. Unlike traditional SR, Eureqa addresses the relationship among all parameters. We chose Eureqa because it is a machine-defined model form approach which can automatically discover power models whose mathematical forms are similar to that generated by human-defined model form approaches. With CSR, application developers can simply and quickly build a power profile (power model) for each smartphone being tested, especially for a smartphone which includes complex subsystems, such as GPU and CPUs. Our main contributions in this paper are as follows:

—We characterize two major behaviors of power consumption in smartphones, i.e., nonlinear and ASP consumption behaviors, which significantly cause the estimation errors.

—To the best of our knowledge, we are the first to use the clustering algorithm method, Affinity Propagation, and Eureqa to improve the accuracy of estimated power consumption of smartphone subsystems.

—We investigate the impacts of these two power consumption behaviors, nonlinear and ASP consumption s, on the real applications running on two different generations of smartphone devices, Nexus S (single CPU core) and Galaxy S4 (Exynos 5 Octa CPU cores).

The remainder of this paper is organized as follows: Section 2 provides the background to our works, the briefs of Affinity propagation and Eureqa; Section 3 gives the problem statement and definition; Section 4 presents our CSR approach; Sections 5 and 6 show the experimental setup and experimental results, respectively, and Section 7 concludes this work.

2 Background

In this section, we describe smartphone subsystems including workload statistics and the aspects of power consumption behaviors, such as ASP and NLP.

2.1 Smartphone Subsystems

A smartphone device is comprised of several hardware subsystems, such as the CPU, GPU, screen, 3G interface, Wi-Fi interface, GPS interface, and so on. Each subsystem is associated with a variety of features. In the scope of this paper, a CPU is associated with four parameters: utilization, frequency, total time duration that a CPU stays in the idle state per second (CPU idle time), and total number that a CPU enters the idle state per second (CPU idle entries). Each subsystem operates in various operating states. Each operating state is represented as a vector storing all parameters values of a subsystem (workload statistics) and the associated power consumption. For example, the workload statistics of busy CPU usage can be presented by four parameters, i.e., utilization = 100%, frequency = 1000 MHz, idle time = 0 ms, idle entries = 0, and the associated power consumption = 600 mW.

2.2 Power Consumption Behaviors

ASP is defined as the consumed power, which is uncorrelated with its corresponding workload statistics. In this paper, we classify ASP into two types:

2.2.1 Predictable power

Predictable power is the ASP where occurrences can be determined in advance, e.g., tail power which is the power consumption that still resides on a subsystem associating with low utilization, e.g., GPS , Wi-Fi [8], and 3G [13] . Fig. 1 shows the high tail power of 3G occurring during the FACH

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2018.2832173, IEEE Transactions on Sustainable Computing

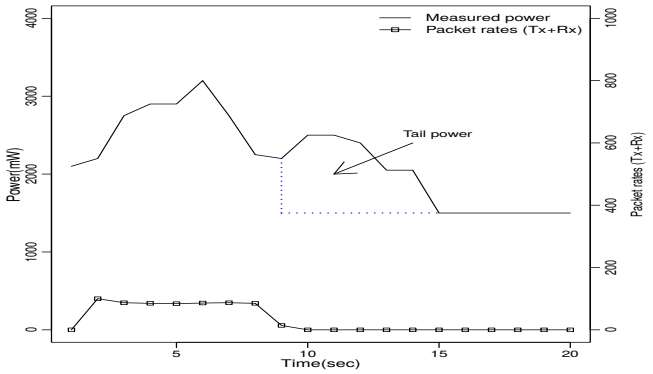


Fig. 1: Asynchronous power behavior: Predictable power such as tail power in 3G.

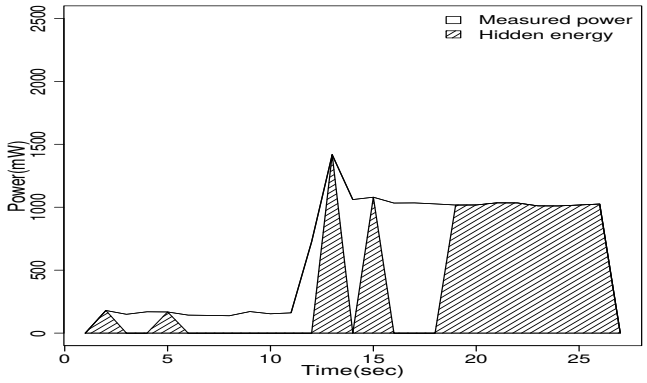


Fig. 2: Asynchronous power behavior: Unpredictable power such as hidden power in GPU.

(Forward Access Channel) state of Radio Resource Control (RRC) protocol of about 6 seconds, i.e., between 9 to 15 sec. In more detail, FACH is the cellular transmission state where smartphones share transmission channels with other phones to reduce the battery power consumption when there is not much traffic to transmit.

2.2.2 Unredictable power

Unpredictable power is the ASP whose occurrences are difficult to determine in advance, namely the hidden power. Fig. 2 shows an example of hidden power, detected by CSR in our experiment, occurring on GPU. Most of the current works have proposed solutions which handle predictable power only, but not unpredictable power. Pathak et al. [8], for example, proposed an instrumentation-based power modeling technique which probes the smartphone system calls and application frameworks to capture tail power of network HW components, but the GPU power remains hidden. Cao et al. [9] also took only CPU and networks into account, but ignores GPU hidden power usage caused by activities such as web-based games. Maghazeh et al. [14] sampled GPU workloads with low sample rates and built a power model with linear regression, which is also not enough to detect GPU hidden power. Although an instrumentation-based technique is more accurate than a utilization-based technique [15], it requires significant development time and system knowledge to handle smartphone software systems, especially for commercial smartphones. In particular, it is a laborious task to apply the instrumentation-based technique to a subsystem such as GPU because of the lack of OS support for GPU abstractions [16].

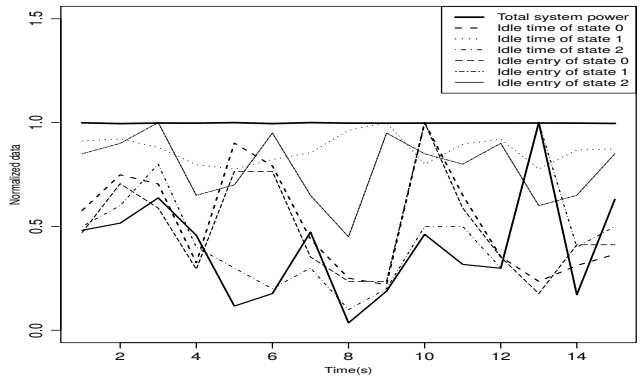


Fig. 3: Nonlinear power consumption behavior of the CPU idle time and entry in state C0-C2 of the CPU core0 on Galaxy S4.

Nonlinear power is defined as the power which is not linearly proportional to workload statistics. For example, Fig. 3 illustrates the normalized power consumption of a CPU to show the nonlinearity of the three levels of CPU idle time and entries. Furthermore, the power nonlinearity also appears for pixel colors on an AMOLED screen, i.e., the HW component showing the highest power consumption ratio compared to other HW components [17] [18] [19]. To model the NLP of CPU idle states, Zhang et al. [10] proposed a weighted linear model to fit the power consumption of multi-core CPU idle states. For AMOLED screens, Radhika et al. [17] and Xu et al. [18] proposed an exponential power model to fit the subset of pixel colors to the associated measured power. To model the NLP of GPU, Ma et al. [7] proposed the statistical method which selects 5 out of 39 GPU parameters to build a Support Vector Regression (SVR). Another statistical method used is a tree-based random forest to build a GPU power estimation model [19]. However, most of the existing works for the GPU power models are based on the studies of a desktop GPU such as Nvidia. Also, the proposed solutions of Zhang et al. [10], Radhika et al. [17], and Xu et al. [18] are all examples of a human-defined model form, whereas those of Ma et al. [7] and Chen et al. [20] are not.

2.3 Affinity Propagation Clustering

The Affinity Propagation (AP) clustering algorithm [11] is based on the process of message passing between data samples i and j. Every data sample is considered as an exemplar, the data that is the center of each cluster. Exemplar continue exchanging messages, responsibility and availability, with one another until a good set of exemplars and corresponding clusters emerges. The responsibility message, $r(i,j)$, is sent from data sample i to data sample j, a candidate exemplar, reflecting the accumulated evidence for how appropriate data sample j is to serve as the exemplar of data sample i. Meanwhile, the availability message, $a(i,j)$,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2018.2832173, IEEE Transactions on Sustainable Computing

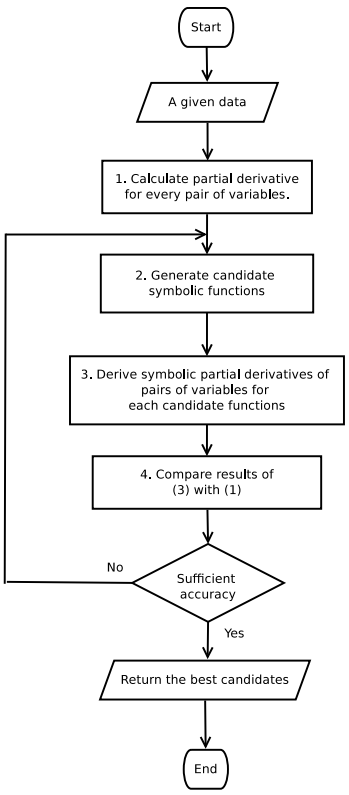
is sent from a data sample j, a candidate exemplar, to the data sample i, reflecting the accumulated evidence for how appropriate it would be for data sample i to choose data sample j as its exemplar. Finally, a set of clusters $\{c_1, c_2, ..., c_n\}$ is selected to maximize the fitness function E, where $E = \sum_{i=1}^{n} s(i, j)$ and s is the similarity function of i and j. The reason we use AP is because it does not require a specified number of clusters in advance. This property is useful for detecting the unknown numbers of ASP occurrence, especially for hidden power.

2.4 Eureqa Symbolic Regression

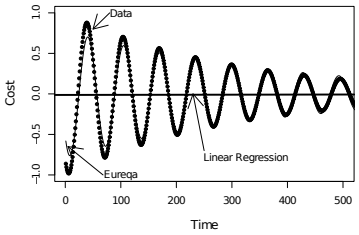
Symbolic regression (SR) is a method for searching mathematical equations from given data samples. Unlike traditional linear and nonlinear regression methods that fit parameters to an equation of a given model form, SR searches both the parameters and equation forms concurrently. The result is represented as a tree structure composed of inner nodes containing the mathematical operators $(+, -, \times, \div, \sin, \cos)$ and outer nodes containing either subsystem parameters or a constant node containing either subsystem parameters or a constant value. Unlike traditional SR whose output model forms are not similar for the same input data trained at different training times, because of its random nature, Eureqa [12] can generate a mathematical model form which is similar to every training.

Fig. 4a shows the process of Eureqa operations. In step 1, Eureqa first works by calculating partial derivatives between variables from given data. In step 2, it then generates candidate symbolic functions that do accurately describe the behaviors of the given data. In step 3, Eureqa derives symbolic partial derivatives of the pairs of variables for each candidate functions. In step 4, the results of step 3 are compared with that of step 1. If the best candidates are not satisfied, step 2, 3, and 4 are iteratively processed until the best candidates are found (see [12] for more details).

Fig. 4b illustrates the example of applying Eureqa to find an equation which describes the behavior of swinging pendulum. Based on the Eureqa's operation, which uses the partial derivative as a key of model searching, Eureqa can produce a list of equations, trading them off between accuracy and simplicity. The list of equations thus generated allows users to have various choices for picking the most suitable equation, e.g., the number 2 in Fig. 4b, which prevents overfitting and underfitting. Unlike traditional linear and nonlinear regressions which fit parameters to an equation of a given form, e.g., the linear regression just give us $f(t) = 0.02$ as shown as number 4 in Fig. 4b, Eureqa can find both the parameters and the form of equations at the same time.



(a) The process of Eureqa symbolic regression.



1. $f(t) = 1.24 \times exp(0.19 - (0.36 \times t)) \times sin((9.76 \times t) \times exp(0.19 - (0.36 \times t)))$
2. $f(t) = exp(0.34 - (0.34 \times t)) \times sin((9.64 \times t) - 5.46)$
3. $f(t) = 0.26 \times sin(0.42 + (9.77 \times t))$
4. $f(t) = 0.02$

(b) Example of Eureqa modeling and a list of equations.

Fig. 4: Eureqa symbolic regression

3 Problem Statement and Definition

This section first introduces the basic definition of smartphone subsystem power estimations, followed by formal definitions of the main problem and subproblems I and II.

3.1 Basic Definitions

The notations used in this paper are as follows. Let $S = s^1, ..., s^M$ denote a set of M subsystems. Each subsystem $s^i \in S$ is characterized by N_i parameters, where s^i_j denote parameter j of the subsystem i. Each subsystem s^i has been trained in different operating states (a 'trained state' for short). Each trained state k of the subsystem i, denoted by $r^i_k = (s^i_{1,k}, ..., s^i_{N^i,k})$, is a vector that stores N^i parameter workloads $s^i_{j,k}$. Each trained state r^i_k is also paired with total system power p^i_k, and is denoted by a training sample $v^i_k = (r^i_k, p^i_k)$. Let $V^i = \{v^i_1, ..., v^i_r\}$ denote a set of Z trained states v^i_k where $1 \leq k \leq Z$. The set V^i is used for building a power model M^i, which represents the relationship between r^i_k and p^i_k. To build an efficient power model M^i, it needs to take ASP and NLP into account. Table I lists all notations and their definitions.

Definition 1. ASP. Given two data samples $v^i_k \in V_i$ and $v^i_l \in V_i$ where $k \neq l$. If the trained state r^i_k is similar to r^i_l ,but the total system power p^i_k is not similar to the total

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2018.2832173, IEEE Transactions on Sustainable Computing

TABLE 1: Notations and their definitions

Notation	Definition
s^i	The subsystem i.
s^i_j	The parameter j of subsystem i.
$s^i_{j,k}$	The workload of parameter j of subsystem i at a trained state k.
r^i_k	The trained state k of subsystem i.
p^i_k	Total system power of subsystem i at trained state k.
v^i_k	The pair of a trained state r^i_k and p^i_k.
V^i	The set of Z trained state v^i_k.
M^i	A power model for subsystem i.
a^i_k	Asynchronous power consumption of a trained state k of subsystem i.
b^i_k	$p^i_k - a^i_k$.
W^i	The set of Z trained state $w^i_k = (r^i_k, b^i_k)$.
A^i	Asynchronous power table for subsystem i.
Y^i	A mathematical equation for subsystem i.
X^i	In estimation, the set of T data samples of subsystem i. Note the data samples do not include power information.
$x^i_{j,t}$	In estimation, the workload of parameter j of subsystem i at time slot t.
x^i_j	In estimation, the vector of all parameters j of subsystem i at time slot t.

system power p^i_l. It refers that either p^i_k or p^i_l includes asynchronous power a^i_k or a^i_l, respectively.

Definition 2. NLP. The nonlinear behavior between the trained states r^i_k and their associated power b^i_k, where $b^i_k = p^i_k - a^i_k$ is the synchronous power. Since the nonlinearity, b^i_k is nonlinear power.

3.2 Problem Statement

The problem statement is as follows. Given a set of training samples V^i, find a power model M^i, which comprises of a mathematical equation Y^i and an ASP table A^i, the table which contains the information of ASP, so that the estimated total system power obtained from the sum of all subsystems $s^i \in S$, under a given workload statistics, is approximately equal to the measured total system power obtained from an external power meter. To find Y^i and A^i, we need to solve two subproblems below.

Subproblem 1. ASP problem. Given a set $V^i = \{v^i_1, ..., v^i_z\}$ where $v^i_k = (r^i_k, p^i_k)$, find ASP a^i_k, where $a^i_k \leq p^i_k$ for each $v^i_k \in V^i$.

Subproblem 2. NLP problem. Given a set $W^i = \{(r^i_k, b^i_k)\}$, find Y^i which is the best fit between the trained states r^i_k and the associated power consumption b^i_k.

4 Clustering and Symbolic Regression

In this section we describe the design of CSR, which has three components: (1) ASP analysis, (2) NLP modeling, and (3) subsystem power estimation, as shown in Fig. 5. The first two components are for training the data samples while the third estimates the power consumption of the subsystems.

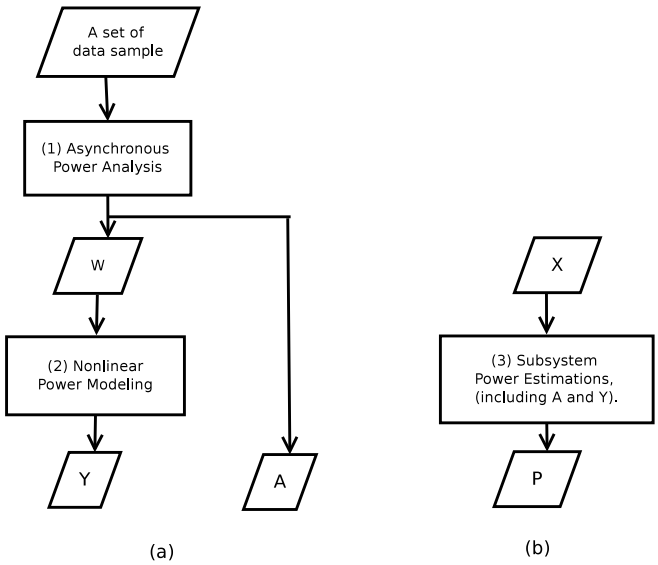


Fig. 5: Workflow diagram of CSR components, (a) ASP analysis and Nonlinear power modeling, and (b) Subsystem power estimation.

4.1 ASP Analysis Component

This component uses Algorithm I, which determines whether a given set V^i contains ASP or not. The algorithm is based on the assumption that if any trained states in V^i are similar, they would be associated with similar amounts of power consumption p^i_k in V^i. If not, there exists some trained states r^i_k which correlate with ASP. Note that this assumption may be too strong when the number of collected parameters is small. We thus collects as many parameters as possible to reduce the errors caused by such an assumption.

To more clearly understand how Algorithm I works, it is illustrated by a simple example as shown in Fig. 6. In Fig. 6(a), a set of data sample $V^i = \{v^i_1, ..., v^i_8\}$, where $v^i_k = (r^i_k, p^i_k)$ is given. We assume that all trained states r^i_k are divided into two groups, based on the similarity values. The first group contains r^i_k, where $k = 1, 2$, and 3, whereas the second group contains r^i_k, where $k = 4, 5, 6, 7$, and 8. Also, all power values p^i_k are divided into two groups. The first group contains p^i_k, where $k = 1, 2, 3, 4$, and 5, whereas the second group contains p^i_k, where $k = 6, 7$, and 8. Based on the assumption mentioned above, it can thus be seen that p^i_4 and p^i_5 contain ASP.

To find p^i_4 and p^i_5 by Algorithm I, we start by partitioning $\forall v^i_k \in V^i$ based on the similarity function and AP clustering (lines 2-3 in Algorithm I). The similarity function, $sim = exp(-(d/w)^r)$, where d is a distance between data, $w = 1$ is a radius, $r = 2$ is an exponent. More details of this function use can be found in [21]. All data samples $v^i_k \in V^i$ are partitioned into two clusters, including a cluster $C_1 = \{v^i_1, v^i_2, v^i_3, v^i_4, v^i_5\}$ and $C_2 = \{v^i_6, v^i_7, v^i_8\}$. Next, all data samples $v^i_k \in V^i$ are clustered again with the same similarity function and AP clustering, but using trained states $\forall r^i_k$ only (lines 4-5 in Algorithm I). Therefore, all data samples $v^i_k \in V^i$ are partitioned into two clusters including $D_1 = \{v^i_1, v^i_2, v^i_3\}$ and $D_2 = \{v^i_4, v^i_5, v^i_6, v^i_7, v^i_8\}$. Finally, the group of clusters C and D are compared in order to find ASP. Our assumption is that all members in D_y should be in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2018.2832173, IEEE Transactions on Sustainable Computing

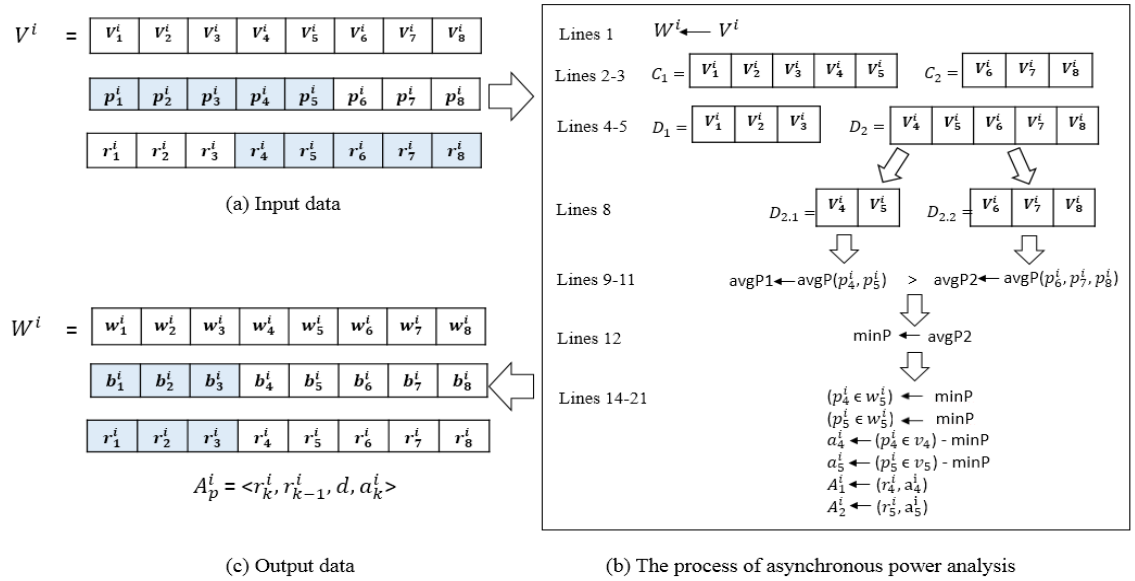


Fig. 6: Asynchronous power analysis.

the same cluster of any C_x. If not, some of members of D_y contain ASP.

In our example, the algorithm will find that $\forall v_k^i \in D_2$ are not in any same cluster, unlike $\forall v_k^i \in D_1$ which are in the same cluster C_1. Hence, it can be concluded that there exist ASP on some $p_k^i \in D_2$. Next, $\forall v_k^i \in D_2$ are partitioned into subclusters in order to find ASP (line 8 in Algorithm I). The number of subclusters depends on the number of clusters C_x in which each $v_k^i \in D_2$ is. In this example, the cluster D_2 is partitioned into two subclusters, i.e., $D_{2,1} = \{v_4^i, v_5^i\}$ and $D_{2,2} = \{v_6^i, v_7^i, v_8^i\}$, because v_4^i and v_5^i are in C_1, and v_6^i, v_7^i, v_8^i are in C_2. In lines 9-11, the algorithm computes the average power of all members in $D_{2,1}$ and $D_{2,2}$, which are $avgP(D_{2,1})$ and $avgP(D_{2,2})$, respectively. These power averages are used to assess which subcluster $D_{y,d}$ contains ASP. Based on these criteria, all subclusters $D_{y,d}$ which do not contain the minimum average power contains ASP. In this example, Fig. 6(b) (lines 9-12) shows that $D_{2,1} = \{v_4^i, v_5^i\}$ contains ASP $a_k^i = avgP1 - avgP2$. Finally, Algorithm I (lines 13-19) produces an ASP table $A^i = \{A_0^i, ..., A_p^i, ..., A_q^i\}$, where $A_p^i =< r_k^i, r_{k-1}^i, d, a_k^i >$ is a row of the table including r_k^i and r_{k-1}^i the pair of the training states triggering the ASP, i.e., r_k^i contains asynchronous power if its previous training state is r_{k-1}^i, d is the asynchronous duration, and a_k^i is the amount of ASP. Finally, a set $W^i = \{(r_1^i, b_1^i), ..., (r_8^i, b_8^i)\}$ is created, where the power b_4^i and b_5^i are modified as $b_4^i = p_4^i - a_4^i$ and $b_5^i = p_5^i - a_4^i$ respectively. Finally, Algorithm I (line 22) returns W^i and A^i, as shown in Fig. 6(c), which are later used by the NLP modeling and subsystem power estimation components, respectively.

Algorithm 1: ASYNCHRONOUS POWER ANALYSIS

Input: A set V^i.
Output: A set W^i //An asynchronous power table A^i.

1 $s1,\ s2 \leftarrow 2D\ matrix;\ C,\ D\ and\ A \leftarrow \emptyset;\ W^i \leftarrow V^i;$
2 $s1 \leftarrow\ sim(\forall v_k^i \in V^i);$
3 $C \leftarrow AP(s1);$ //Clustering
4 $s2 \leftarrow\ sim((\forall r_k^i \in v_k^i) \in V_i);$
5 $D \leftarrow AP(s2);$ //Clustering
6 **foreach** $D_y \in D$ **do**
7 **if** $\forall v_k^i \in D_y$ *are not in the same cluster* C_x **then**
8 $F \leftarrow partition(\forall v_k^i \in D_y);$
9 **foreach** $D_{y,d} \in F$ **do**
10 //Add to list
11 $avgP.add(\frac{\sum((\forall p_k^i \in v_k^i) \in D_{y,d})}{b});$
12 $minP \leftarrow argmin\{avgP\};$
13 // Number of members of $D_{y,d}$
14 $d \leftarrow |D_{y,d}|;$
15 **foreach** $(p_k^i \in v_k^i) \in D_{y,d}$ **do**
16 $a_k^i \leftarrow abs(p_k^i - minP);$
17 $w_k^i \leftarrow (r_k^i,\ minP);$
18 $A^i.add(< r_k^i,\ r_{k-1}^i,\ d,\ a_k^i >);$
19 **return** $W^i,\ A^i;$

4.2 Nonlinear Power Modeling Component

Algorithm II is applied to this component. The algorithm uses Eureqa to produce a mathematical equation representing the relationship between trained states $r_k^i$and the associated synchronous power b_k^i.

Algorithm II requires two input parameters, i.e., the set W_i obtained from Algorithm I, and the target expression E which is a string expression that guides Eureqa the type of model to search for. For instance, a target expression "$y = f(x_1, x_2, ..., x_n)$" is an equation where y is modeled as a function of variables $x_1, x_2, ..., x_n$ and $y = f_1()x_1 + f_2()x_2 +, ..., +f_n()x_n$ is an equation where $f_i()$ is the coefficients of a variable x_i. Algorithm II (line 1) works by initially defining the empty lists L and H. Eureqa then applies the two input parameters to build the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2018.2832173, IEEE Transactions on Sustainable Computing

Algorithm 2: NONLINEAR POWER MODEL GENERATION

input: A set W^i
A target expression $E \leftarrow$ "$y = f(x_1, \ldots, x_k)$"
Output: An equation Y^i.
1 $L, H, \leftarrow$ *An empty list*;
2 $L \leftarrow Eureqa.build(W^i, E)$;
3 $M \leftarrow$ "$y = \forall x_i \in L$";
4 $H \leftarrow Eureqa.build(W^i, M)$;
5 $G \leftarrow \{\forall h_i \in H \mid MAE(h_i)$ *is* 10% *larger than* $MAE(\frac{argmax}{h_i \in H} Com(h_i))$
6 $Y^i \leftarrow \frac{argmax}{h_i \in H} Com(g_i)$
7 **return** Y^i;

equation (line 2). Since Eureqa uses Symbolic Regression (SR) to build an equation, it has to set termination criteria, i.e., a number of generations and a stability value. The stability value is used for measuring the sensitivity analysis of all parameters to find a set of parameters most effecting the subsystem power consumption. After completing, the construction process, Eureqa returns a list of the most impact subsystems parameters L. Next, the target expression E is modified by replacing the existing variables with a new set of variables from the list L, denoted as M (line 3). Eureqa is again run by applying W_i and M, and then returns a list of candidate equations H (line 4). The list H, a collection of candidate equations, are ranked based on the trade-off between the accuracy (fit) and complexity (size) of an equation. The accuracy is measured by using the Mean Absolute Error (MAE) metric and the complexity is the size of an equation. We defined the process of choosing an equation Y^i from H in two steps: (1) picking a group of candidates whose MAEs are 10% larger than the MAE of the greatest complexity, and (2) selecting the equation which has the greatest complexity from the group we picked at the first step. To create a generic model, we propose the picking process which picks an equation that has the least complexity but highest accuracy.

Fig. 7 is illustrative of the process of Algorithm II. At step 1, let W^i be a set of pairs of trained states and power consumption $w_k^i = (r_k^i, b_k^i)$ of GPU. The GPU subsystem is composed of 19 parameters, $x_1, ..., x_{19}$. The set W^i and the target expression E are submitted to Eureqa that applies sensitivity analysis to generate a set of GPU's parameters most effecting its power consumption. In this example, the sensitivity analysis generates the four most impact parameters, x_1, x_5, x_8, x_{12}. At step 3, the target expression E is replaced by "$Y = f(x_1, x_5, x_8, x_{12})$". At step 4, the expression E and W_i are submitted to Eureqa to build a model again, but without applying the sensitivity analysis. At step 5, Eureqa returns the results of a list of equations which are a trade-off between accuracy and complexity. Finally, at step 6, Algorithm II uses the picking criteria mentioned above to pick the right equation Y^i that is the equation whose complexity and MAE are 15 and 3, respectively, as shown as the red spot in step 5.

4.3 Subsystem Power Estimation Component

This component uses Algorithm III, as well as the ASP table A^i and the equation Y^i, obtained from Algorithm I and II, respectively, to estimate the total subsystem power

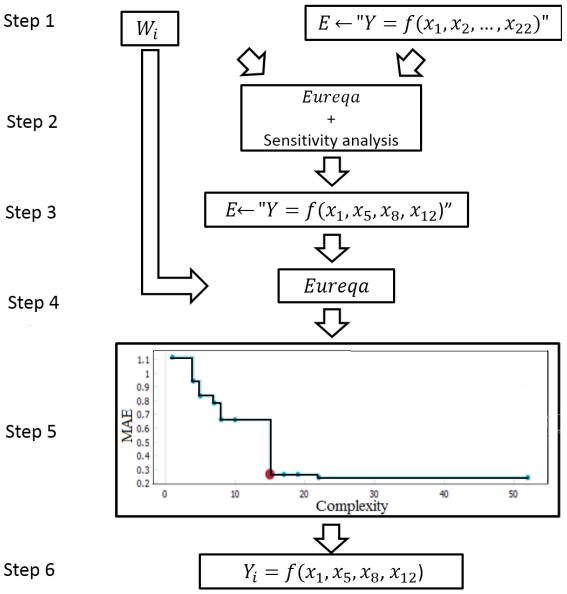


Fig. 7: Nonlinear power modeling.

of the subsystem s_i, given a set of T data samples X^i. Note that data samples X^i only includes the workload statistics of the subsystems, but not their power information. Algorithm III (line 1) works by initially setting an empty list P. Next, for each data sample $x_t^i \in X^i$, Algorithm III (line 3) checks whether x_t associates with ASP or not. This works by measuring the similarity, sim (line 4), between x_t and $r_k^i \in A_p^i$ and x_{t-1} with $r_{k-1}^i \in A_p^i$. If the similarity is greater than zero, it means x_t^i associates with ASP, we then estimate the total system power P_t of x_t^i, by applying x_t^i to the equation Y^i plus $a_k^i \in A_p^i$, and adding asynchronous duration d with t to the variable len. Alternatively, P_t is estimated by applying x_t^i with Y^i only. If x_t^i associates with ASP, we then continue checking the duration $d \in A_p^i$ as shown in lines 7–13. In this iterative process, the similarity between x_t^i and x_{t-1}^i is measured. If both data samples are similar, then the total system power consumption P_t^i is also estimated by applying x_t^i with the process, as shown in line 5. Alternatively, x_t^i is checked with the other $A_p^i \in A^i$. Finally, this component returns a set of T total system power P^i associated with the data samples X^i.

5 EXPERIMENTAL SETUP

In this section, we first describe the hardware and software experimental setup. We then elaborate all subsystems and their training process.

5.1 Hardware and Software Setup

5.1.1 Hardware setup

The hardware test bed consisted of a host computer, a Monsoon power monitor [22], and two DUTs. The host computer was a normal desktop computer with a 3.10GHz Intel Core i3-2100 processor, 6GB of RAM, and 64-bit Windows 7. The Monsoon power monitor, which sampled with rates at 5 kHz, was used to measure the total system power. To test the efficiency of the proposed technique, we ran our

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2018.2832173, IEEE Transactions on Sustainable Computing

Algorithm 3: SUBSYSTEM POWER ESTIMATION

Input: A set of T data sample $X^i = \{x_1, \ldots, x_r\}$.
Output: A set of T power estimation $P^i = \{p_1, \ldots, p_r\}$.

$P^i \leftarrow$ () //An empty list;
for $t = 1, \ldots, |X_i|$ **do**
  foreach $A_p^i \in A^i$ **do**
    if $sim(x_t^i, r_k^i \in A_p^i) \wedge sim(x_{t-1}^i, r_{k-1}^i \in A_p^i)$ **then**
      $len \leftarrow t + d \in A_p^i$;
      $p_t \leftarrow Y_i(x_t^i) + a_k^i$;
      while $t < len$ **do**
        if $sim(x_i, x_{i+1})$ **then**
          $P_t \leftarrow Y_i(x_{t+1}) + a_k^i$;
          $t++$;
        else
          $t--$;
          break;
    else
      $P_t \leftarrow Y_i(x_t)$
return P^i;

experiments on the old and new DUTs, Nexus S and Galaxy S4 (S4 for short). More details of both DUTs, including the target subsystems, CPU, screen, GPU, audio, GPS, 3G, and Wi-Fi are listed in Table II.

5.1.2 Software setup

The software tools consisted of Eureqa desktop version 0.99.8 [23], a statistical tool, R [24], with the Apcluster package [25], a system call tracing tool, Strace [26], and our training application running on both DUTs. To validate the accuracy of CSR, we tested it with four real applications with several scenarios. The applications included GPU benchmark, Google Maps, Firefox web, Firefox Youtube, Chrome web, and Chrome Youtube. For the Eureqa parameter setup, for each training process, we ran the evolutionary process until the number of generations reached 30,000 or a stability value of more than 80%. It took around 4 minutes to complete each training process.

5.2 Subsystem Training Process

To acquire subsystem workload statistics, we created two applications, operating and monitoring applications [27], working on a DUT side. The operating application put a target subsystem into different operating states, while the monitoring application periodically collected the workload statistics of the training subsystems. To synchronize the subsystem workload statistics with its associated power measurements, we used the instantaneous power caused by suddenly turning on and off the screen brightness as the synchronization point. After completing the training process, we stored the collected workload statistics of subsystems in a DUTs storage, and the power trace in the host computer. For each operating state of a subsystem, we repeatedly tested it five times and then measured its average. The details of each subsystem training process are as follows:

1) CPU. We disabled other subsystems when training the CPU. However, since GPU is on the same System-on-Chip as CPU, we also monitored the workload statistics of GPU. The training parameters of CPU were set as described in Zhang et al. [9], for training CPU idle state. While, it was simple to train a single CPU core on Nexus S, it was an intensive task to train 8 CPU cores on S4. The 8 CPU cores, big.LITTLE [28], were partitioned into 2 groups: one group for 4 big cores and the other group for 4 little cores. Each big core had a range of frequencies of between 800 and 1600 MHz, and each little core had a range of frequencies of between 200 and 600 MHz. Although the S4's CPU was composed of 8 cores, but only four cores, or one group, were active at a time, because of limitations of the current scheduler technology, In-kernel switcher [28] at the time. Developers can in fact only view 4 logical CPU cores, core0, core1, core2, and core3. The OS kernel allows developers to turn off core1 to core3, but not core0. Thus, to train the S4's CPU, we started by training core0 alone by disabling the other cores. We next trained core1 by enabling it and let it operate along with core0. The power consumption of core1 is then estimated by subtraction. We subsequently trained core2 and core3 using the same procedure

TABLE 2: The subsystems of Nexus S and Galaxy S4

Sub system	Nexus		Galaxy S4	
	Parameter	Range	Parameter	Range
CPU	$util_0$(%)	$1-100$	$util_0, ..., util_7$	$1-100$
	$freq_0$(MHz)	{200, ..., 1000}	$freq_0, ..., freq_7$	{200, ..., 1600}
	it_0 (idle time (ms))	≥ 0	$it_{0,0}, ..., it_{7,0}$	≥ 0
			$it_{0,1}, ..., it_{7,1}$	≥ 0
			$it_{0,2}, ..., it_{7,2}$	≥ 0
	ie_0 (idle time (ms))	≥ 0	$ie_{0,0}, ..., ie_{7,0}$	≥ 0
			$ie_{0,1}, ..., ie_{7,1}$	≥ 0
			$ie_{0,2}, ..., ie_{7,2}$	≥ 0
Screen	brightness	$0-255$	bright	$0-255$
			red	$0-255$
			green	$0-255$
			blue	$0-255$
GPU (show the most impact four parameters)	tal	≥ 0	tal	$\geq$
	fps	≥ 0	fps	≥ 0
	usseccpp	≥ 0	usseccpp	≥ 0
	gtt3d	≥ 0	gtt3d	≥ 0
Audio	volume	[0, 1]	volume	[0, 1]
GPS	on	[0, 1]	on	[0, 1]
Wi-Fi	on	[0, 1]	on	[0, 1]
	channel	{11, 36, 48, 54}	channel	{11, 36, 48, 54}
	packet_rate	≥ 0	packet_rate	≥ 0
3G	on	[0, 1]	on	[0, 1]
	packet_rate	≥ 0	packet_rate	≥ 0

tal : tile accelerator utilization
fps : frame per second description
usseccpp : Universal Scalable Shader Engine clock cycle per pixel
gtt3d : GPU task time 3D utilization

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2018.2832173, IEEE Transactions on Sustainable Computing

as for core1.

2) Screen. It was simple to train the screen of Nexus S because it was a Super LCD [29], where its power consumption is affected only by brightness levels. However, for the screen of S4, we had to address the red, green, and blue pixel colors for each brightness level, starting from 0 to 255 with increments of 25. Therefore, at each brightness level, the 24-bit color values, 8 bit for each color, were varied from 0 (black) to 255 (white). Since an OS kernel does not provide the pixel color data, for real application testing we used another android application, SCR screen record [30], to record the pixel colors of a whole screen, while the real application was running. The pixel color data was then saved as a MP4 file within a DUT, and was later processed on the host computer. To reduce the power consumption overhead caused by the SCR screen record app, it was set to record the screen at eight frames per second. Moreover, to reduce the time spent on pixel processing on the host computer, we processed only 135x240 of the total of 1080x1920 pixels. The power consumption of the screen was obtained by subtracting the power consumption of CPU from the total system power. We also observed that there were no GPU operations while pixel colors were being trained by our training application.
3) GPU. 19 GPU parameters were trained and it was an intensive task to control all combinations of these parameters. We thus used the GPU benchmark, 0xbench [31], to stress test the GPU. 0xbench consists of 2D training apps, such as canvas, shape, and image drawing, 3D training program, such as Cube and Teapot rotation, and other miscellaneous apps, e.g., Math, VM, Native, etc. The power consumption of GPU was obtained by subtracting the power consumption of CPU and screen.
4) GPS. We used the GPS Test application to train only GPS on and off states to capture its ASP consumption. The power consumption of GPS was estimated by subtracting CPU and screen power consumption. 5. Audio. We trained the audio subsystem by running the built-in music player, Apollo, and only maximum and minimum volume were trained. Its power consumption was estimated by subtracting CPU power because the screen was off.
5) Wi-Fi. We set the host computer as a server that connected with a router TP-Link TL-WR1043ND. We developed a client-server application, as described in PowerTutor [5], to train the Wi-Fi workload statistics. To reduce the variability of the experiment, we controlled Wi-Fi channel rates at 11, 36, 48, 54, and 72 Mbps, while the files with different sizes were exchanged between the DUT to the server at each channel rate. We found that the Wi-Fi subsystem of our DUTs resulted in a short duration of ASP consumption, i.e., less than 1 second. We thus ignored the asynchronous analysis for the Wi-Fi subsystem.
6) 3G. We experimented with 3G similar to the Wi-Fi experiment. However, we estimated its power consumption by transferring multiple files with various sizes between the DUT and a server over FTP.

6 Experimental Results

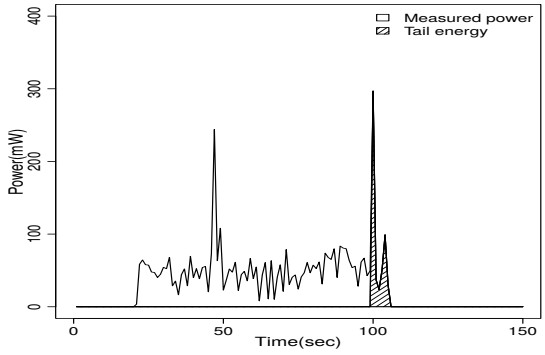
6.1 ASP Detection

We give the results of CSR with reference to detecting ASP as it occurred on GPS, GPU, and 3G. We ignored Wi-Fi as its duration of ASP was trivial, i.e., about 1 second. Fig. 8(a) shows the ASP of GPS, which lasts for about 5 seconds after GPS is disabled. We compared the CSRs results with the results of the instrumentation-based approach, which uses Strace to determine the operating states of GPS. Our experiment found that the accuracy of CSR to detect ASP is closed to the accuracy of the instrumentation-based approach. However, CSR can automatically obtain the time duration of ASP, whereas the instrumentation-based approach requires detecting the time duration of asynchronous power manually. Moreover, we found that a sampling rate of 1 Hz was sufficient for detecting ASP of GPS. Fig. 8(b) shows ASP of GPU on S4. CSR revealed several portions of ASP occurring on GPU. After determining the GPU workload statistics that are correlated with ASP, we found that most of ASP on GPU is hidden power. For example, between 12 and 14 seconds all captured workload statistics are similar, but the associated power consumption changes instantaneously. Moreover, we found that the ASP of GPU on Nexus S was caused by the GPUs parameters correlated with negative values, e.g., the negative values of Universal Scalable Shader Engine (USSE) load stall and load pixel. We also found that at least the sampling rate at 10 Hz was suitable for detecting ASP of GPU. Since it was difficult to instrument the GPUs system calls for the comparison purposes, we, therefore, validated the accuracy of CSR for identifying the ASP of GPU with the real applications. Fig. 8(c) shows the comparison between the techniques with and without detecting the ASP on 3G. The techniques for detecting the ASP include CSR and system call (FSM), whereas the technique without detecting ASP is LM. It can be seen that the techniques with detecting ASP can reduce more errors than the technique without detecting ASP.

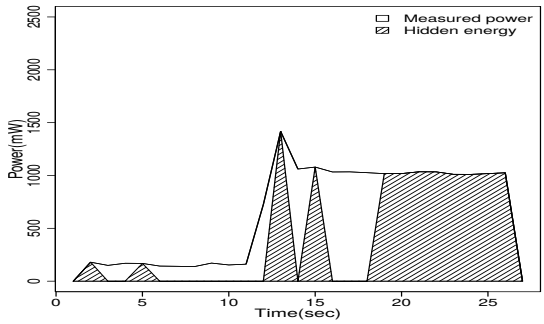
6.2 Nonlinear Power Models

To validate the efficacy of CSR in terms of using a machinedefined model forms for discovering NLP models, we compare the forms of models generated by CSR with the ones generated by using the human-defined model forms. To do so, we built a power model of the Nexus S CPU by using the Weighted Linear Model (WLM) described in Zhang et al. [10], whose form was defined by a human. Table III shows that, without human intervention, CSR can determine a model whose form is very similar to the ones produced by WLM. Moreover, the Mean Absolute Error (MAE) of both of these are approximately the same, 17.95 and 18.49, respectively. We also built a CPU power model generated by using GP, a traditional symbolic regression. As shown in Table III, the model form obtained from the GP approach is completely different from the one obtained from WLM as well as CSR. The difference is because the random nature of genetic programming which means the model forms built by GP are different for each model built,

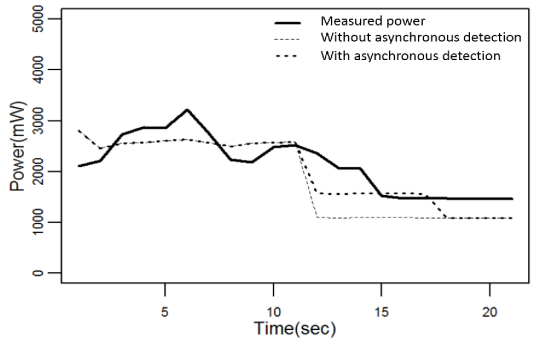
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2018.2832173, IEEE Transactions on Sustainable Computing



(a) GPS tail energy.



(b) GPU hidden energy.



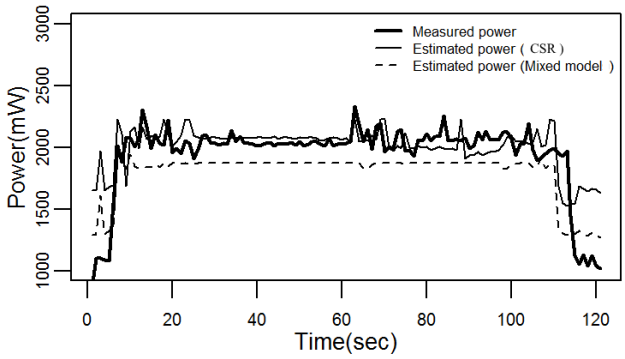
(c) 3G tail energy.

Fig. 8: Asynchronous power detection on (a) GPS, (b) GPU, and (c) 3G.

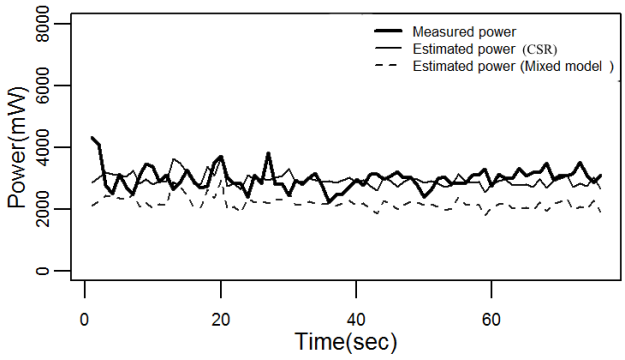
even though the input data is the same, whereas CSR can produce very similar forms of models for each model building. To show the accuracy, we compared the accuracy of the NLP models built for GPU of Galaxy S4, the most complex hardware subsystem. The results showed that the MAE of the GPU power model built by LM is very high. The MAE of LM and CSR are significantly different, at 8387.01 and 956.67, respectively. Furthermore, we also compared the results of CSR with another nonlinear model building method, SVR [7]. The results show that the error of CSR is less than that of SVR, at about 15% on GPU.

6.3 Real Application Validation

To evaluate the accuracy of CSR-based power estimation, we validated it on five well-known applications (seven scenarios) and then compared its results with those of the mixed model. In this paper we refer the mixed model as a



(a) GPS tail energy.



(b) GPU hidden energy.

Fig. 9: Asynchronous power detection.

list of subsystem power modeling techniques which gave the best results for each subsystem we tested. For the mixed model, we applied WLM for CPU, an Exponential model for AMOLED, SVR for GPU, and LR to the other subsystems.We also used Mean Absolute Percentage Error (MAPE) as the error metric for evaluation.

Table IV shows the MAPE of CSR and the mixed model, tested with the five applications running on two DUTs. On Nexus S, the average MAPE of CSR was about 10.58%, whereas that of the mixed model was about 13.85%, i.e., a 23.61% improvement of CSR over the mixed model. We found that the accuracy of estimated GPU power consumption has the most impact on that of total system power estimation, and the estimated power consumption of the other major subsystems, such as CPU, screen, and Wi-Fi was similar. As shown in Fig 9(a), CSR is more accurate than the mixed model for estimating total system power consumption of surfing whole CNN website for about 2 minutes testing on Chrome with Wi-Fi and 3G. The higher accuracy results from CSR being able to correctly capture the power consumption behavior of GPU, whereas SVR cannot perform this. On S4, the average MAPE of CSR was about 23.41%, whereas that of the mixed model was about 40.75%, i.e., about 42.55% improvement. Fig. 9(b) shows that CSR is capable of capturing accurate GPU power consumption behavior by improving around 70.05on GPU bench application. The improvement is because of the capability of ASP detection of CSR on GPU.

It is worth noting that all MAPEs in this work were sig-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSUSC.2018.2832173, IEEE Transactions on Sustainable Computing

TABLE 3: GPU and CPU power models of Nexus S and Galaxy S4 built by linear, GP, and CSR methods

DUT	Subsystem	Method	Power model	MAE
Nexus S	CPU	CSR	$262 + (2.05 \times util) + (0.35 \times freq) + (\frac{it}{(it - 32.12 - (32.12 \times ie))})$	17.95
		WLM	$268.3 + (2.12 \times util) + (0.34 \times freq) + (-0.4662 \times \frac{it}{1+ie})$	18.49
		GP	$\frac{it}{2.23} + (\frac{util}{0.21} + util + (\frac{util}{0.21} + (-3.32 + \frac{ie}{(1+it)})))$	-
Galaxy S4	GPU	LM	$(188.84 \times tal) + (-7.17 \times fps) - (-1371.69 \times usseccpp) + 1828.59$	8387.01
		CSR	$603.76 + (1702.98 \times gtt3d^2) + (147.69 \times gtt3d^4) - fps - (1370 \times gtt3d) - (857.65 \times gtt3d^3)$	956.67
		SVR	SVM-Type: eps-regression, SVM-Kernel: radial, cost: 1, gamma: 0.05263158, epsilon: 0.1	1125.61

TABLE 4: Mean Absolute Percentage Error (MAPE) of power estimation on real applications

DUT	Technique	GPU bench	Google maps	Chrome web (Wi-Fi)	Chrome web (3G)	Firefox web (Wi-Fi)	Firefox web (3G)
Nexus S	Mixed model	8.15	15.61	13.49	19.90	7.64	18.07
	CSR	7.97	13.72	9.52	15.86	6.30	13.00
Galaxy S4	Mixed model	38.49	34.83	24.31	77.49	30.40	58.27
	CSR	11.62	20.88	23.03	18.86	28.47	25.68

nificantly high, because of the power consumption overhead caused by using SCR screen record application periodically capturing all pixel colors of the screen when real applications were being tested. In our experiment, it showed that the SCR app consumes about 19% of the total system power consumption.

7 Conclusion

In this paper we develop a Clustering and Symbolic Regression (CSR) power estimation method for smartphone subsystems. CSR improves the accuracy of subsystem power estimation by investigating nonlinear and asynchronous power consumption behaviors. With AP clustering, CSR can efficiently capture ASP consumption behaviors on the subsystems for which the instrumentation-based methods are complicated to apply. Without a predefined form of a power model, CSR can discover a power model which not only fits the NLP consumption behavior but also discover the correct relation between parameters as similarly defined by a human, e.g., the discovery of power model of CPU with three parameters, i.e., utilization, frequencies, and idle states. We suggest that the ability of automatically build a model without requiring the knowledge of the relationship between subsystems parameters will play an important role in the modeling methodology for embedded systems in the future. We conducted experiments to evaluate the accuracy of CSR by comparing it with various models, LM, WLM, GP, and SVR. Our evaluation shows that CSR can reduce the MAPE of total system power estimates between 23.61% and 42.55%, from the results of various real applications running on two different smartphones. CSR is however still limited for analyzing the asynchronous power of some HW subsystems which requires the instrumentation to obtain their workload statistics, e.g., I/O device drivers. We will take this into account in future work.

Acknowledgments

This work was done when E. Rattagan was with National Chiao Tung University.

References

[1] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury, "Detecting energy bugs and hotspots in mobile apps," in *Proc. SIGSOFT*, 2014, pp. 588-598.
[2] Android power profile [Online], Available: https://source.android.com/devices/tech/power/index.html
[3] M. Dong, and Z. Lin, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proc. MobiSys*, 2011, pp. 335-348.
[4] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures." in *Proc MICRO*, 2009, pp. 168178.
[5] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proc CODES/ISSS*, 2010, pp. 105114.
[6] M. B. Kjrgaard and H. Blunck, Unsupervised Power Profiling for Mobile Devices, in *Proc. MobiQuitous*, 2012, pp. 138-149.
[7] X. Ma, M. Dong, L. Zhong and Z. Deng, "Statistical Power Consumption Analysis and Modeling for GPU-Based Computing," in *Proc. HotPower*, 2009.
[8] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proc. EuroSys*, 2011, pp. 153168.
[9] Y. Cao, J. Nejati, P. Maguluri, A. Balasubramanian, and A. Gandhi, Analyzing the Power Consumption of the Mobile Page Load. *In SIGMETRICS Perform. Eval*. Rev. 44, 1, 2016, pp. 369-370.

[10] Y. F. Zhang, X. D. Wang, X. Z. Liu, Y. X. Liu, L. Zhuang, and F. Zhao, "Towards better CPU power management on multicore smartphones," in *Proc. HotPower*, 2013.

[11] B. J. Frey and D. Dueck, "Clustering by Passing Messages Between Data Points," *Science*, 2007.

[12] M. Schmidt and H. Lipson, "Distilling Free-Form Natural Laws from Experimental Data," *Science*, 2009, pp. 8185.

[13] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. "Energy consumption in mobile phones: a measurement study and implications for network applications," *Proc. IMC*, 2009, pp. 280293.

[14] A. Maghazeh, U. D. Bordoloi, M. Villani, P. Eles, Z. Peng, "Perception-aware power management for mobile games via dynamic resolution scaling," in *Proc. of the IEEE/ACM International Conference on Computer-Aided Design ICCAD '15*, pp. 613-620, 2015.

[15] C. Yoon, D. Kim, W. Jung, and C. Kang, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring," in *Proc. of USENIX ATC 12*, 2012.

[16] C. J., Rossbach, J. Currey, and E. Witchel, "Operating Systems must support GPU abstractions," in *Proc. HotOS*, 2011.

[17] M. Radhika, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proc. Mobicom*, 2012.

[18] F.Y. Xu, Y.X. Liu, Q. Li, and Y.G. Zhang, "V-edge: fast self-constructive power modeling of smartphones based on battery voltage dynamics," in *Proc. USENIX NSDI*, 2013, pp. 43-56.

[19] X. Chen, N. Ding, A. Jindal, Y. Charlie Hu, M. Gupta, and R. Vannithamby, "Smartphone Energy Drain in the Wild: Analysis and Implications," in *Proc. of SigMetrics* 2015.

[20] J. M. Chen, B. Li, Y. Zhang, L. Peng, and J.- K. Peir, "Statistical GPU power analysis using tree-based methods," in *Proc. IGCC*, 2011, pp.1-6.

[21] Sim function. expSimMat [Online]. Available: http://cran.irsn.fr/web/packages/apcluster/apcluster.pdf

[22] Monsoon power monitor. A power meter [Online]. Available: http://www.monsoon.com/LabEquipment/PowerMonitor.

[23] Eureqa. Symbolic regression tool [Online]. Available: http://creativemachines.cornell.edu/eureqa.

[24] R. Statistical software tool [Online]. Available: http://www.r-project.org.

[25] APCluster. Affinity Propagation clustering package [Online]. Available: cran.r-project.org/web/packages/APCluster/index.html.

[26] Strace. A system call tracer [Online]. Available: http://benno.id.au/blog/2007/11/18/android-runtime-strace

[27] SRC tool [Online]. Available: https://github.com/pokekarat/SRC_tool-master

[28] Linaro. In Kernel Switcher. [Online]. Available: https://events.linuxfoundation.org/images/stories/slides/elc2013_poirier.pdf.

[29] Super LCD [Online]. Available: http://en.wikipedia.org/wiki/Nexus_S.

[30] SCR screen record [Online]. Available: https://play.google.com/store/apps/details?id=com.iwobanas.screenrecorder.free.

[31] $0xbench$. Android benchmark suite [Online]. Available: https://code.google.com/p/0xbench/.

Ekarat Rattagan received the MS degree in Information Technology from King Mongkuts University of Technology Thonburi (KMUTT), Bangkok, Thailand, in 2003, and the Ph.D. degree in Electrical Engineering and Computer Science from National Chiao Tung University, Hsinchu, Taiwan, in 2016. He is currently a lecturer in the Faculty of Information Science and Technology at Mahanakorn University of Technology, Bangkok, Thailand. His research interests include mobile embedded systems.

Ying-Dar Lin is a Distinguished Professor of computer science at National Chiao Tung University (NCTU), Taiwan. He received his Ph.D. in computer science from the University of California at Los Angeles (UCLA) in 1993. He was a visiting scholar at Cisco Systems in San Jose during 20072008, and the CEO at Telecom Technology Center, Taiwan, during 2010-2011. From August 2017, he has been jointly appointed as the Vice President of National Applied Research Labs (NARLabs). Since 2002, he has been the founder and director of Network Benchmarking Lab (NBL, www.nbl.org.tw), which reviews network products with real traffic and has been an approved test lab of the Open Networking Foundation (ONF) since July 2014. He also cofounded L7 Networks Inc. in 2002, later acquired by D-Link Corp, and O'Prueba Inc. in 2018. His research interests include network security, wireless communications, and network softwarization. His work on multi-hop cellular was the first along this line, and has been cited over 850 times and standardized into IEEE 802.11s, IEEE 802.15.5, IEEE 802.16j, and 3GPP LTE-Advanced. He is an IEEE Fellow (class of 2013), IEEE Distinguished Lecturer (20142017), ONF Research Associate, and received in 2017 Research Excellence Award and K. T. Li Breakthrough Award. He has served or is serving on the editorial boards of several IEEE journals and magazines, and is the Editor-in-Chief of IEEE Communications Surveys and Tutorials (COMST). He published a textbook, Computer Networks: An Open Source Approach (www.mhhe.com/lin), with Ren-Hung Hwang and Fred Baker (McGraw-Hill, 2011).

Yuan-Cheng Lai received his Ph.D. degree in the Department of Computer and Information Science from National Chiao Tung University in 1997. He joined the faculty of the Department of Information Management at National Taiwan University of Science and Technology in August 2001 and has been a professor since February 2008. His research interests include performance analysis, protocol design, wireless networks, and and network security.

Edward T.-H. Chu received the Ph.D. degree in computer science in 2010 from the Department of Computer Science at National Tsing Hua University, Hsinchu, Taiwan. He has more than 4 years work-experience in the industry, where he worked on embedded software and owns a Chinese patent. He was a visiting scholar at Purdue University in 2009. He joined the Department of Electronic and Computer Science Information Engineering at National Yunlin University of Science and Technology, Taiwan, as an assistant professor in 2010 and has become an associate professor in 2015. His research interests include embedded systems and real-time operating systems.

Kate Ching-Ju Lin received the B.S. degree from National Tsing Hua University, Taiwan, in 2003 and the Ph.D. degree from National Taiwan University, Taiwan, in 2009. She has been a visiting scholar in CSAIL, MIT, in 2007. She is now an associate professor in Computer Science at National Chiao Tung University. Her current research interests include wireless systems, RF-based sensing and visible light communication.