Fine-Grained Energy Estimation and Optimization of Embedded Operating Systems

Xia Zhao [1, 2], Yao Guo [2], Hua Wang [2], Xiangqun Chen [2]
1. College of Computer Science and Technology, Beijing Technology and Business University, Beijing 100037, China
2. Key laboratory of High Confidence Software Technologies (Ministry of Education), Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China;

***Abstract*—Embedded operating systems (EOS) manage the resources of the system and control device operations, and play an important role on optimizing system energy consumption. This paper proposes a new approach to estimate and optimize the energy consumption of the EOS and the applications at a fine-grained level. The approach is based on a micro-architectural power model and a new estimation model for operating system energy consumption. We apply the approach to an Intel Strong-Arm architecture platform running embedded Linux 2.4.18, analyzing its energy characteristics and also trying to optimize energy of the applications on it based on the analyzing results. The experiments demonstrate that the approach can identify energy consumption of fine-grained software components correctly and be used to optimize the energy consumption of EOS and applications.**

I. INTRODUCTION

Due to the severely limited energy supply and the growing capability for executing resource-intensive tasks, reducing the power dissipation of systems becomes a primary design target for mobile computers and portable embedded systems.[1-2]. In order to improve the energy efficiency of the system, evaluating energy consumption of various components in the system from the perspective of software is very important.

Software energy consumption is defined as the energy consumption of the system components while the programs running on processor and during memory accesses. In order to estimate energy consumption of EOS and applications during the early design stage of embedded systems, software energy consumption estimation techniques based on power model and estimation model needs to be established beforehand.

Recently, there are a lot of research on software energy consumption model [3] and evaluation approach on embedded systems [4-5] and user-level applications [6]. Detailed and fine-grained energy consumption characteristics estimation and analysis of embedded operating system is lacking.

In most of multi-tasks embedded operating systems, system calls are application developers-perceived interfaces of EOS. Besides system calls, there are many other kernel services and routines running underlying the interface, such as interrupt handlers, exception handlers, and schedule routines. These routines are invoked by random events that are invisible from outside. It becomes difficult to estimate energy consumption of EOS in a finer granularity using the existing methods. Furthermore, it is much more difficult to optimize EOS and applications energy consumption without fine-grained energy estimation ability.

In this paper, we propose a new approach to estimate and optimize the energy consumption of EOS and applications efficiently. We propose a estimation model for operating system energy consumption based on a micro-architectural power model and EOS functionality and structural characteristics. The proposed approach is able to estimate energy consumption of system calls and kernel execution path separately. The experiments demonstrate that the approach can estimate energy consumption of fine-grained software components correctly and be used to optimize the energy consumption of an EOS and applications running on it.

The remainder of the paper is organized as follows. Section 2 presents the proposed approach, including the fine-grained EOS energy consumption estimation model. Section 3 describes experiment results. Section 4 shows an energy optimization example. Section 5 reviews related work. Finally, Section 6 concludes the paper.

II. FINE-GRAINED ENERGY ESTIMATION OF AN EOS

A. Overview

An overview of the proposed estimation framework is illustrated in Fig. 1. Within the framework are three major components: a full-system instruction level simulator executing the OS and applications; a micro-architectural power simulator estimating cycle-accurate power dissipation of instructions, and a software energy analyzer integrating multiple-granularity software energy consumption. The inputs to the framework are an executable binary OS kernel image file and a root file system involving user-level test programs.

The full system instruction simulator simulates functionalities of microprocessor and peripherally components, and can run an unmodified embedded OS. It outputs run-time instruction and address streams to the micro-architectural power simulator through a message queue. The micro-architectural power simulator simulates operations of micro-architectural components of pipelines and memory access. During instructions execution in the pipeline, the simulator calculating per-cycle power

This work was supported by the National High Technology Development 863 Program of China under Grant No. 2007AA010304 and No. 2007AA01Z462.

978-0-7695-3288-2/08 $25.00 © 2008 IEEE
DOI 10.1109/ICESS.Symposia.2008.58

dissipation of micro-architectural components based on their power model [6]. It sends cycle-accurate power consumption of instructions and corresponding instruction addresses to the software energy analyzer. The latter treats a run-time operating system as a set of logical units consisting of atomic functions, routines, services, and execution paths. It builds run-time function call tree on the fly by analyzing instruction-address sequence and symbol information of OS; and then calculates multiple-granularity software energy consumption of OS based on software energy estimation-model.

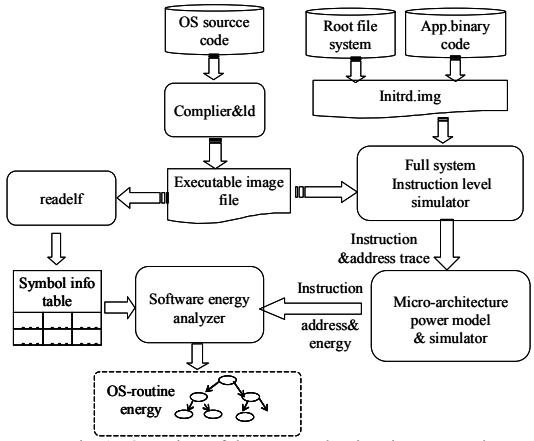

Fig. 1. Overview of the proposed estimation approach

The power model accounts for effects such as branch delays, pipeline stalls, control-flow mispredictions, cache misses, etc. It arranges the micro-architectural components into four classes: datapath, cache, clock, and memory. The power dissipation of datapath, cache and clock are estimated based on the model proposed by Austin etc. [7]. The off-chip memory system has different states during the execution of the system, and the active state consumes major parts of the energy. We use the current in active state to model the energy consumption of the memory.

The energy consumption of an instruction flowing along the pipeline is calculated by the following equation:

$$e_i = e_{datapath} + e_{cache} + e_{sram} + e_{clock} + e_{misc} \quad (1)$$

Where, $e_{datapath}$ is the energy dissipation of datapath in pipelines, e_{cache} is that of the Cache, e_{sram} is that of the TLB, e_{clock} is that of the clock circuits, and e_{misc} is the that of miscellaneous logical units.

B. Fine-grained EOS energy consumption estimation model

To accurately estimate EOS energy consumption at a finer granularity, we propose a new energy consumption estimation model that treats the run-time EOS as a set of logical units organized in hierarchy based on their functionality and structural characteristics.

We define four types of units in EOS, namely atomic function, routine, service, and execution path. An *atomic function* is an atomic unit of EOS consisting of sequential instructions, without invoking other functions. A *routine* is a collection of instructions and atomic functions calling each other. A *service* is a kind of special routine with special entry and executes through the hardware trap mechanism. Services can be divided into three classes: system call, exception service and interrupt service. A *kernel execution path* describes a process beginning with the first instruction in kernel mode, and ending at the return instruction to user mode, which is compose of routines and services.

1) Energy consumption of atomic functions

Energy consumed by an atomic function is defined as the sum of per-cycle power dissipation of sequential instructions belonging to the atomic function. The per-cycle power dissipation is the sum of power dissipation of every processor components in one cycle. In order to ensure the validity of the function energy calculation by accumulating the per-cycle power dissipation of instructions within an atomic function, we exploit the combinability of the summarize operation by pipeline simulation.

Another issue of atomic function energy estimation is to determine whether an instruction belongs to a certain function. We propose an approach of comparing instructions address with function symbol information to identify sub-function invocation and return. We set up a function symbol table, and build run-time function call tree to identify the complex program structure.

2) Energy consumption of routines and services

A routine can be represented as a runtime function call tree consisting of functions with invocation relations among them. The energy consumption of the routine is defined as the energy consumption of the root node, noted as $E_{Routine} = E_{f_0}$, which is calculated by recursive aggregation of the energy consumption of its sub-tree nodes. Given the routine's function call tree consisting of node $f_0, f_1, ..., f_n$, every node denotes a function, f_0 is the root node. Let E_{f_i} is the energy consumption of the function f_i, I_{f_i} is the instruction set of this function, $f_{j1}, f_{j2}, ..., f_{jm}$ $(i < j_1 < j_2 < ... < j_m < n)$ are its sub-functions, and e_i is the per-cycle energy consumption of instructions; then, the energy consumption of any node on this tree is the sum of energy consumption of its sub-function and all instructions belong to this node directly, expressed as follow:

$$E_{f_i} = \begin{cases} \sum_{i \in I_{f_i}} e_i & (f_i \text{ has no sub-functions}) \\ \sum_{i \in I_{f_i}} e_i + \sum_{k=1}^{m} E_{f_k} & (f_i \text{ has sub-functions } f_k) \end{cases} \quad (2)$$

The estimation algorithm of EOS routine and service energy consumption is as Figure 2.

The energy estimation of a kernel service is similar as a routine. The only difference is that kernel service has special program entries, and needs to be identified among instruction streams.

3) Energy consumption of execution paths

The existing energy estimation approaches [8-9] calculate energy consumption of a system call with energy consumption of the process from the first instruction into kernel mode to the last instruction return to user mode. In fact, they misestimated it by assigning the energy consumption

throughout the process as the energy consumption of the system call.

```
Repeat:
   Fetch the instructions address and energy from the buffer queue;
   Identify the function which the instruction belongs to; assign the function name to variable func;
   IF (is system call or exception entry) THEN new a tree;
   Assign the current node to pointer variable cur_node;
   IF (func equals cur_node) THEN
       Add the energy of the instruction to the current node, CONTINUE;
   Search the function tree which the instruction belongs to
   IF (an node is found) THEN
       Assign the current node to pointer variable pre_node;
       Assign the found node to pointer variable cur_node;
       Add the pre_node energy to cur_node;
       CONTINUE;
   New a tree, and assign new node to cur_node;
```

Fig.2. Pseudo Code of Estimation Algorithm of EOS Routine and Service Energy Consumption

In order to estimate and analyze the kernel energy consumption accurately, we define this process as *kernel execution path*, which including system call services and other services. Therefore, kernel services are classified into two types. An *explicit service* is defined as a system call service which is invoked by user programs, and only consists of routines implementing the system call function, such as *sys_read* etc. An *implicit service* is defined as a kernel service which is transparent to user programs, and has no direct relations with the current execution, such as interrupt and exception handlers.

Hence, we distinguish the system call service from system call process. The frontier is an explicit service, while the latter is a kernel execution path. Energy estimation of kernel execution path can be done in two cases. 1) *simple execution path*: This execution path is composed of a system call service without interrupted, and its energy consumption $E_{execpath}$ is the energy of the service $E_{service}$. 2) *complex execution path:* when the explicit service is interrupted, the execution path is a forest of function call trees including explicit services and implicit services throughout the process. The energy of the path is the sum of energy of all the sub-trees. The expression is as follow:

$$E_{execpath} = \begin{cases} E_{explict\_service} & (\text{simple execution path}) \\ E_{explict\_service} + E_{implict\_service} + E_{Routine} & (\text{complex execution path}) \end{cases} \quad (3)$$

III. Experimental Results and Discussions

We implement an energy consumption estimation tool, which simulates an embedded system featuring a StrongARM micro-processor, and estimates the energy consumption of the embedded operating system kernel ARMLinux 2.4.18. The tool consists of an improved full system instruction simulator based on Skyeye [10], an improved micro-architectural power simulator [7] and a software energy analyzer. Table 1 shows the configuration of the micro-architectural model used for our experiments.

Some of our test programs come from real embedded system application software, while others are designed to exercise specific OS routines and services. Table 2 shows the list of benchmarks used.

TABLE I
Micro-architecture model

Parameter	Value
Feature Size	0.18um
V_{dd}	1.5V
Frequency	200MH
Fetch/Issue/Retire Width	2 (inorder)
RUU/LSQ size	4
L1 I-Cache	16KB (32B cache line, 2-way assoc.)
L1 D-Cache	8KB (32B cache line, 32-way assoc.)
TLB (full assoc) entries	32

The output of the estimation tool is a hierarchical list of function call trees with properties of routine on each tree node. As shown in Figure 4, every property includes the routine name and energy consumption calculated on micro-architectural datapath and memory access separately.

TABLE II
Benchmarks

Interactive programs	Non-interactive programs
cat, ls, ln, echo, more, du	rm, mount, cp, gzip, gunzip, tar
ifconfig, ftpget, telnet	netstat
ps, whoami	kill, chgrp,
Msg client, Shm client	Msg server, Shm server, pipes

A. Energy consumption of kernel routines and services

We classify the test programs into two categories: interactive and non-interactive. From the data shown in Fig. 3, we observed the following characteristics of the kernel energy consumption.

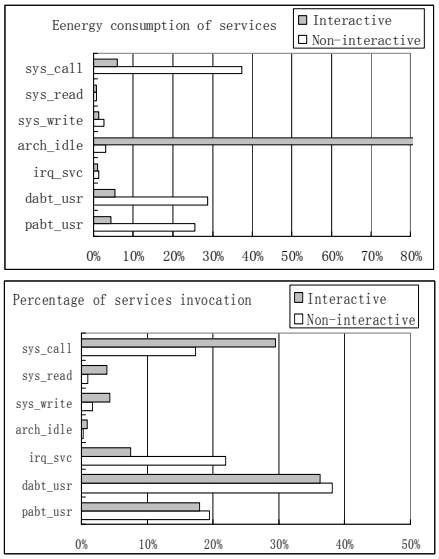

Fig.3. Distribution of Number and Energy Consumption of Routines and Services

1) The percentage of energy consumption of processor idle (arch_idle) in interactive applications is much higher than that of in non-interactive applications. The former is 81% in overall energy consumption, while the latter is only 3%, because of the notable processor idle waiting time during

human-computer interaction.

2) The number of invocation and energy consumption of system calls (sys_xxx) in two scenarios are quite different. In the interactive scenario, the number of system calls makes up 37% of overall kernel invocations, which is 17% higher than in non-interactive applications. However, the energy consumption of the former is only 8%, 34% lower than the latter's.

3) The number and energy consumption of exception (page fault exception and prefetch exception) handling archive a significant proportion to overall kernel energy consumption in both scenarios. The explanation for this is that on the StrongARM platform, ARMLinux kernel manages the memory with virtual memory technology, which loads codes and data of the process into memory through on-demand paging and prefetch paging mechanisms.

It can be concluded that processor idle is one of main sources of energy consumption in interactive system, while the system calls and exception handling are major sources of energy consumption in non-interactive system. So optimization of these kernel routines and services is one way to reduce energy consumption of EOS.

B. Functionality vs. energy consumption of routines and services

The experiment results indicate that energy consumption of kernel routine/service is directly related to the complexity of its functionality. Table 3 gives the results of statistic of energy consumption of some kernel routines/services.

TABLE Ⅲ
ENERGY CONSUMPTION OF SOME ROUTINES AND SERVICES

Name	Counts of executions	Energy consumption（uJ）			
		Means	Min	Max	Std.
sys_getegid	25	4.99	3.52	9.51	1.70
sys_getgid	20	2.17	1.95	2.71	0.18
sys_getpid	19	1.55	0.20	2.17	0.41
__switch_to	110	1.24	1.08	1.78	0.16
__wake_up	427	2.78	0.20	5.97	0.77
sys_brk	147	12.21	4.45	29.60	7.00
sys_fork	24	89.40	73.10	113.73	11.04
sys_execv	24	453.31	372.00	1219.60	164.83
sys_exit	21	138.34	115.81	188.5	19.09
sys_clone	6	43.88	25.18	72.39	16.24
sys_open	44	71.63	29.2	445.94	62.105
sys_write	291	80.97	38.72	204.94	32.71
sys_read	266	44.74	8.1	355	40.78
sys_close	66	78.44	8.02	330.58	93.59
__dabt_svc	39	46.04	32.98	165.15	20.53
__pabt_user	1211	34.27	191.27	402.69	37.36
__irq_svc	244	34.12	9.5	86.57	12.41

1) The routines with simple functionalities have low and stable energy consumption. The mean of energy consumption of the ID operation routines, such as *sys_gerpid,* is not more than 5uJ. The standard deviation of these routines is not more than 2.

2) The kernel routines/services with remarkable energy consumption should be given more attentions. For example, *sys_fork* is used to create new process, it energy consumption is 89.40uJ, which is higher than that of *sys_clone*, which is used to create new thread in the same process. If we can use sys_clone instead of sys_fork to perform concurrent operations, we can reduce the cost of time, memory and energy consumption.

3) The energy consumption of complex services and kernel execution paths are quite different in different scenarios. As shown by Figure 4, the different kernel execution paths with the same system call have distinct function call tree and energy consumption under different conditions.

```
sys_read(micro:17.005,mem:0.000)
    fget(micro: 8.527,mem:0.000)
    generic_file_read(micro: 3.336,mem:0.000)
      do_generic_file_read(micro:2.854,mem:0.000)
        update_atime(micro:1.534,mem:0.000,)
  fput(micro: 1.130,mem:0.000)
)
```

(a) Energy consumption of sys_read in execution path of generic_file_read

```
sys_read(micro:17756.991,mem:0.000,
    fget(micro:3.058,mem:0.000,)
    tty_read(micro:17750.125,mem:0.000,
        read_chan(micro:17749.156,mem:0.000,
            ...
    schedule(micro:2.124,mem:0.000,
      __switch_to(micro:1.200,mem:0.000,)
    cpu_idle(micro: 17720.347,mem:0.000,)
    __irq_svc(micro: 31.609,mem:0.000,
    fput(micro: 1.132,mem:0.000)
)
```

(b) Energy consumption of sys_read in execution path of tty_read

Fig.4. Energy Consumption of Different Execution Paths

C. Energy consumption of system calls

In a multi-tasks concurrent operating system, with implicit services in kernel execution path, some system calls are different from system call processes, and the energy consumptions of them are very different.

TABLE Ⅳ
ENERGY CONSUMPTION OF DIFFERENT PATHS（uJ）

OS service name	tty_read	generic_file_read（from storage）	generic_file_read（from memory）
Execution	1705	123	40
Explicit	41	78	40
interrupt	130	45	0
schedule	10	0	0
Idle process	1476	0	0
Timer interrupt	48	0	0
Proportion of Explicit services	2.46%	63.41%	100%

As shown in Table 4, *sys_read* system call process has three types of kernel execution paths under different cases. The first case is performing *tty_read* to read a user input from keyboard which has some implicit services. The energy consumption of *sys_read* system call process is 1705uJ, while the energy consumption of *sys_read* system call is just 41uJ, holding not more than 3% of that in total process. The second case is performing *generic_file_read* to read a file from the storage and the energy is 123 uJ. The implicit services *dabt_handler* are invocated to load the file from flash device to memory. The third case is performing *generic_file_read* to read some data existing in memory and the energy is 40uJ. The energy consumptions of system call and system call process are the same, because no implicit services are invoked.

The results demonstrate that the implicit services have significant impact on the energy estimation of OS system

calls and applications. With the approach proposed by this paper, we can distinguish energy consumption of a system call from a execution path based on the fine-grained energy consumption information, and can identify key factors of the EOS energy consumption for software energy consumption optimization.

IV. Energy Optimization in an EOS

Based on the above analysis, we show how to optimize an Inter-Processes Communication (IPC) mechanism selection for given programs to reduce energy consumption. The basic objective of optimization is to select energy efficient IPC and minimize the number of IPC, reducing intervention of underlying OS services. The optimization has following properties:

1） *The functionalities of applications remain the same as before;*

2） *The total volume of IPC messages passed between the two processes remains the same as before;*

Consider two chat programs implemented on a multi-processes EOS. As shown in Figure 5, the chat programs consist of two processes, *server* and *client*. The *client* process reads string inputting by user, and passes the string to the *server* process that is waiting on another IPC end. The *server* receives the string and displays it. Three different IPC mechanisms, namely pipe, message queue and sharing memory are implemented respectively. The functionality and the communicated data are the same in each IPC mechanism.

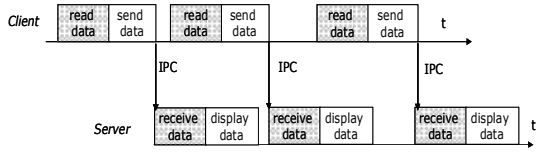

Fig.5. The *server* and *client* of the chat programs

Energy consumptions of three IPC mechanisms in the chat programs are quite different. As shown in Figure 6, message queue tends to be more energy-efficient compared to pipe and sharing memory. Based on the analysis of energy consumption of these IPC mechanisms, we select message queue to implement the data communication in the chat programs. As shown in Fig. 6(b), Energy consumption of message queue mechanism during one time execution of the chat programs is only 2.7mJ, that of pipe mechanism is 6.8mJ, and that of share memory mechanism is 12.9mJ. Hence, the message queue mechanism can reducing 59%energy consumption than pipe mechanism, and 78% than sharing memory mechanism.

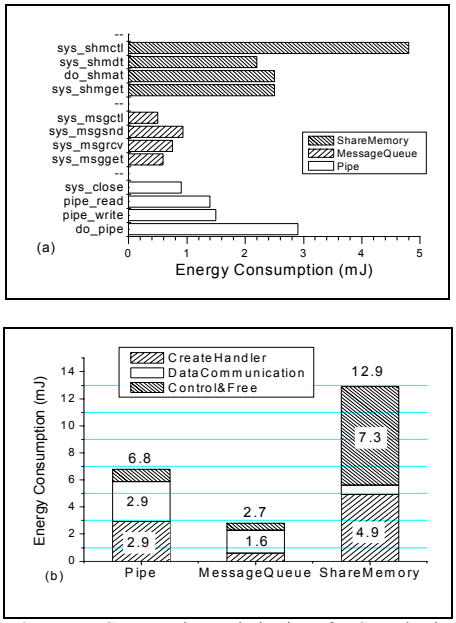

Fig.6. Energy Consumption optimization of IPC mechanisms

The experiment demonstrates that, with our approach and the tool, the detailed energy consumption analysis of services and routines of EOS kernel can be performed, which enables optimizing applications and operating systems more efficiently.

V. Related Works

Tiwari [2] proposed the concept of software energy consumption and a method of instruction-level power modeling in 1994. The objective of software energy consumption research is to study the effects of software on energy consumption of processor and system by mapping hardware energy consumption to software structures and functionalities, and to support energy-aware software design. There have been several methods to estimate or evaluate software energy consumption. Tan et al. developed an energy simulator EMSIM [11]to estimate energy consumption of an embedded system software based on instruction-level energy model. Flinn *et al.* [12]and Acquaviva *et al.* [9]profile energy usage of mobile applications and embedded operating system in a wearable device, using hardware instrumentation to measure current level. This method needs high resolution instrumentation to measure energy consumption of overall embedded system. It is hard to identify the impact of individual services or routines of OS by this method.

Most of the existing architectural level power simulators (e.g. Wattch[3], Power Analyzer[7]) provide cycle-accurate simulation of inner parts of the processors using detailed processor models. But they mainly focus on energy consumption estimation of user-level programs. It is hard for these architectural-level simulators to provide full-system simulation for OS execution and energy estimation.

Analyzing and optimizing the energy consumption of OS and application running on it is another area wherein the characteristics of OS kernel energy consumption have been

examined. Tan *et al.* proposed a software architectural transformation approach [13]to reduce energy consumption of software. Fei *et al.* [14]improved this approach into a source code transformations approach. However, analysis of EOS energy consumption targeting the overall kernel services has not been sufficiently studied.

VI. CONCLUSIONS

In this paper, we propose a new approach of estimating and optimizing the energy consumption of embedded OS (EOS) at a finer granularity. In order to provide fine-grained energy estimation of an EOS, a new software energy estimation model is presented based on micro-architectural power model and EOS functionality and structural characteristics. We perform experiments on an Intel Strong-Arm architecture running embedded Linux 2.4.18, which demonstrates that the approach can identify energy consumption of fine-grained software components correctly and it can be used to optimize the energy consumption of an EOS and the applications running on it.

The proposed approaches are not relied on a specific architecture or operating system thus can be used to build a new operating system energy consumption estimation platform if combined with a different instruction simulator and micro-architectural power model.

REFERENCES

[1] T. K. Tan, A. Raghunathan and N. Jha, "Embedded Operating System Energy Analysis and Macro-modeling," in *Proc. Proceedings of the International Conference on Computer Design*, 2002. pp. 517-522.

[2] V. Tiwari, S. Malik and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Transactions on Very Large Scale Integration*, vol.2(4), 1994, pp. 437-444.

[3] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. 27th Annual International Symposium on Computer Architecture*, 2000.

[4] K. Chandra and G. Selim, "A run-time, feedback-based energy estimation model For embedded devices," in *Proc. Proceedings of the 4th international conference Hardware/software codesign and system synthesis*, Seoul, Korea: , 2006.

[5] A. Dunkels, F. sterlind, N. Tsiftes and Z. He, "Software-based On-line Energy Estimation for Sensor Nodes," in *Proc. In Proceedings of the Fourth Workshop on Embedded Networked Sensors (Emnets IV),*, Cork, Ireland: ACM Press, 2007. pp. 23-27.

[6] T. Li and L. K. John, "Operating System Power Minimization through Run-time Processor Resource Adaptation," *Journal of Microprocessors and Microsystems*, vol.30(4), IPC Science and Technology Press, 2006, pp. 173-224.

[7] H. Angus, "A Survey of Economic Problems Awaiting Investigation in British Columbia," *Contributions to Canadian Economics*, vol.2(, 1929, pp. 45-51.

[8] R. P. Dick, G. Lakshminarayana, A. Raghunathan and N. K. Jha, "Power Analysis of Embedded Operating Systems," in *Proc. Proceedings of the 37th conference on Design automation*, Los Alamitos, California: Ieee computer society Press, 2000. pp. 312-315.

[9] A. Acquaviva, L. Benini and B. Ricco, "Energy Characterization of Embedded Real-Time Operating Systems," in *Proc. Workshop on Compilers and Operating Systems for Low Power, COLP'01*, 2001. pp. 13-18.

[10] Y. Chen, *The Analysis and Practice on Open Source Embedded System Software--Based on SkyEye and ARM Developing Platform.*, Beijing: Beihang University Press, 2004.

[11] M. Gottdiener, "Some Limits to the Parsons Revival: Comment on Sciulli," *American Journal of Sociology*, vol.91(3), 1985, pp. 674-677.

[12] J. Flinn and M. Satyanarayanan, "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications.," in *Proc. Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, 1999. pp. 2-10.

[13] T. K. Tan, A. Raghunathan and N. K. Jha, "software architecture transformations：A New Approach to Low Energy Embedded Software," in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, 2003. pp. 11046.

[14] Y. Fei, S. Ravi, A. Raghunathan and N. K. Jha, "Energy-Optimizing Source Code Transformations for OS-driven Embedded Software," in *Proc. Proceedings of the 17th International Conference on VLSI Design (VLSID＇04)*, 2004.