The Journal of Systems and Software 85 (2012) 1757–1769

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

SEProf: A high-level software energy profiling tool for an embedded processor enabling power management functions

Shiao-Li Tsao*, Jian Jhen Chen

Dept. of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, ROC

ARTICLE INFO

Article history:
Received 4 November 2010
Received in revised form 21 January 2012
Accepted 12 March 2012
Available online 31 March 2012

Keywords:
Energy profiling
Power management
Embedded processor
Power consumption

ABSTRACT

Energy efficiency has become one of the most important design issues for embedded systems. To examine the power consumption of an embedded system, an energy profiling tool is highly demanded. Although a number of energy profiling tools have been proposed, they are not directly applicable to the embedded processors with power management functions that are widely utilized in battery-operated embedded systems to reduce power consumption. Hence, this study presents a high-level energy profiling tool, called SEProf, that estimates the energy consumption of an embedded system running multithread software and a multitasking operating system (OS) that supports power management functions. This study implements the proposed SEProf in Linux 2.6.19 and evaluates its performance on an ARM11 MPCore processor. Experimental results demonstrate that the proposed tool can provide accurate energy profiling results with a low profiling overhead.

1. Introduction

Energy efficiency has become one of the most important issues in the design of embedded systems, especially for battery-operated devices such as mobile phones. To evaluate the energy efficiency of an embedded system, accurate energy profiling tools are required. Previous studies on the energy profiling of embedded software adopt measurement-based and model-based approaches. Measurement-based approaches, such as PowerScope (Flinn and Satyanarayanan, 1999), directly measure the power consumption of an embedded system using an oscilloscope or a digital multimeter, and profiling software runs on the target system to collect system events. This approach analyzes the energy consumption of the embedded software by associating measurement results with system events. The types and number of system events recorded by the profiling software and the sampling rate of the metering software restrict the granularity of power consumption analyses of the system, and setting up a high-resolution measurement environment and recording a large number of system events are costly. Synchronizing the measurement data on the meter and the system events on the target is also challenging because the metering and profiling software run on different machines. Furthermore, measurement-based tools usually generate a large amount of measurement data and system logs, and post processing is time consuming.

The other approach to profile the energy consumption of an embedded system is based on power models. Model-based tools estimate the energy consumption of embedded systems by monitoring the occurrences of representative events during system execution, along with the energy weights of these events. Model-based tools can be further classified into low-level model and high-level model approaches. Low-level model approaches consider architecture-level or instruction-level power models for a processor and estimate the power consumption of software via simulation or emulation of software execution. Wattch (Brooks et al., 2000) adopted an architecture-level power model that was integrated into the SimpleScalar simulator (Burger and Austin, 1997). Wattch modeled the power consumption of the primary units of an embedded processor, e.g. functional units and caches, and monitored the number of accesses to these units to estimate the energy consumption of embedded software (Monchiero et al., 2008). Unlike architecture-level power models, instruction-level models (Tiwari et al., 1994; Sinha and Chandrakasan, 2001; Blume et al., 2007a,b) divide a processor's instruction set into a number of classes according to the average power consumption of each instruction execution. These tools can determine the energy consumption information of a program by accumulating the number of executed instructions for each class. Tan et al. proposed EMSIM (Tan et al., 2002, 2003), based on the instruction-level power model presented in (Sinha and Chandrakasan, 2001) to further support per-task and function-level energy estimation in an embedded Linux environment.

* Corresponding author at: Room EC426, No. 1001 University Road, Hsinchu 300, Taiwan, ROC. Tel.: +886 3 5712121x54717.
E-mail address: sltsao@cs.nctu.edu.tw (S.-L. Tsao).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2012.03.027

Although low-level model approaches can provide accurate evaluation results, they require a significant amount of time to collect necessary information for energy estimation. This is particularly true for embedded systems running complex software, such as multithread programs and multitasking operating systems (OSs). High-level model tools estimate the power consumption of software at basic block levels (Tiwari et al., 1994; Tan et al., 2001) or function levels (Qu et al., 2000; Hsu et al., 2007; Li and John, 2003). The power consumption of basic blocks or functions is first determined via direct measurements or low-level power models. The embedded software then runs directly on a target platform, and energy-profiling tools collect execution information of these basic blocks or functions to estimate the power consumption of the software. High-level model approaches make a tradeoff between the profiling accuracy and overhead. Tiwari et al. (1994) built a base energy cost for basic blocks of the target program. The energy consumption of the program can be evaluated by accumulating the number of times that each basic block is executed multiplied by its base energy cost. Another basic-block power analysis was proposed in Tan et al. (2001). This approach groups consecutive basic blocks in the target program together, and derives the energy weight of each group using regression analysis. Qu et al. (2000) presented a function-level power analysis. In this approach, a database, or power data bank, stores the average power and execution times of library functions and basic instructions. This method evaluates the energy consumption of a program through the number of times that each function is invoked multiplied by its average power and execution time recorded in the power data bank. Another function-level power analysis tool proposed in Hsu et al. (2007) is a software energy estimation tool for heterogeneous dual-core processor. This function-level power model measures the average power consumption of different digital signal processing (DSP) algorithms in advance and stores the data in an energy library. The energy consumption of DSP algorithms can be calculated by multiplying the execution time of each DSP algorithm by its average power in the energy library. Li and John (2003) revealed a correlation between instructions per cycle (IPC) and the average power consumption of OS routines, and proposed a linear regression model to estimate the power consumption of OS routines instead of assuming constant power consumption for the evaluation.

Unfortunately, these high-level tools do not consider the power management functions usually supported by modern embedded processors. Embedded processors, and especially those designed for battery-operated devices, are sensitive to power consumption, and provide sophisticated operating modes, voltages, and frequencies (Choi et al., 2005; Isci et al., 2006; Kumar et al., 2003). Operating systems (OSs) can use the power management features of the embedded processors to achieve dynamic power management functions, optimizing the energy efficiency of the embedded system. For example, ARM9, ARM11, Marvell (formerly Intel) XScale, Texas Instruments OMAP, etc. all support dynamic voltage and frequency scaling (DVFS) functions. The Linux kernel starting from version 2.6.0 also provides a CPUfreq subsystem to facilitate DVFS management and control. Without considering the operating modes, voltages, and frequencies of an embedded processor and the dynamic power management functions of an OS, software energy profiling results become inaccurate. As a result, system designers are limited in their ability to evaluate the power management strategies on the embedded system.

This study presents an energy profiling tool called SEProf. The proposed SEProf is similar to a model-based tool since we manipulate the power tables to derive the power consumption of a software. Although, in this paper, the power tables were obtained through a measurement-based tool, the power tables can be also derived by using model-based approaches such as Wattch (Brooks et al., 2000) and EMSIM (Tan et al., 2002, 2003). Compared with measurement-based approaches, the proposed tool reduces the cost and overhead for evaluating the power consumption of an embedded system. The proposed tool also reduces the profiling time significantly when compared with low-level model approaches. The proposed tool is a practical solution for evaluating the power consumption of a complex embedded system with a multitasking OS and multithread programs. Unlike other conventional high-level approaches, our design considers the power management functions of modern embedded processors and OSs such as different idle mode, suspend mode, and dynamic voltage and frequency scaling, etc., and can provide a fast run-time power estimation of a complex embedded system. System designers can thus run the embedded system under different execution environments and run-time parameters for a time period, examine the characteristics of the power consumption of the system, and fine tune dynamic power management strategies of the embedded software. Furthermore, the proposed tool supports energy profiling at different granularities. Therefore, system designers can apply various profiling granularities in different processes and/or functions on an embedded system according to the requirements of the profiling accuracy. The proposed SEProf was implemented in Linux 2.6.19. Experimental results show that the average energy estimation error of using SEProf is less than 2%, and the overhead introduced by SEProf is less than 5%.

The rest of this paper is organized as follows. Section 2 describes the design and implementation of the proposed tool, SEProf. Section 3 presents experimental results and a case study based on an ARM11 MPCore processor. Section 4 concludes this study.

2. Design and implementation of SEProf

Fig. 1 provides an overview of the proposed energy-profiling tool, *SEProf*. Before profiling embedded software, we have to construct a power table database for the target embedded processor in Step 1. This *power table database* is a collection of power tables that SEProf uses to estimate the power consumption of the processor, and it can be built via measurement tools or low-level model tools. In this study, we used a direct measurement tool to build the power tables in SEProf. If low-level model tools are used, the power table can be constructed by running the programs on an instruction-level energy simulator, and the power tables for the process and its functions can be obtained. It is important to note that different embedded systems may share certain common processes and/or functions, such as an OS and libraries; the power tables of these common processes and functions can be reused when evaluating the energy consumption of other embedded systems based on the same embedded processor.

A *power table* has a number of entries, and each entry denotes the average power consumption required for the processor to execute multiple programs, a multithread program, or a code block of a program. A *code block* represents a sequence of instructions, such as a basic block, a function, or a sequence of basic blocks in OS or applications. The granularity of entries in a power table indicating the profiling granularity influences the profiling accuracy and the overhead, and the profiling granularity is a configurable parameter in SEProf. SEProf is designed to support different granularities for profiling the energy consumption of embedded software. If system designers require a fast estimation of the energy consumption of an embedded system and can tolerate certain estimation errors, they can specify a per-system power table or per-process power tables in SEProf to achieve fast evaluation. Upon identifying the major processes that consume the most energy, they can specify per-function power tables for the major processes and conduct further detailed examination. System designers can make a

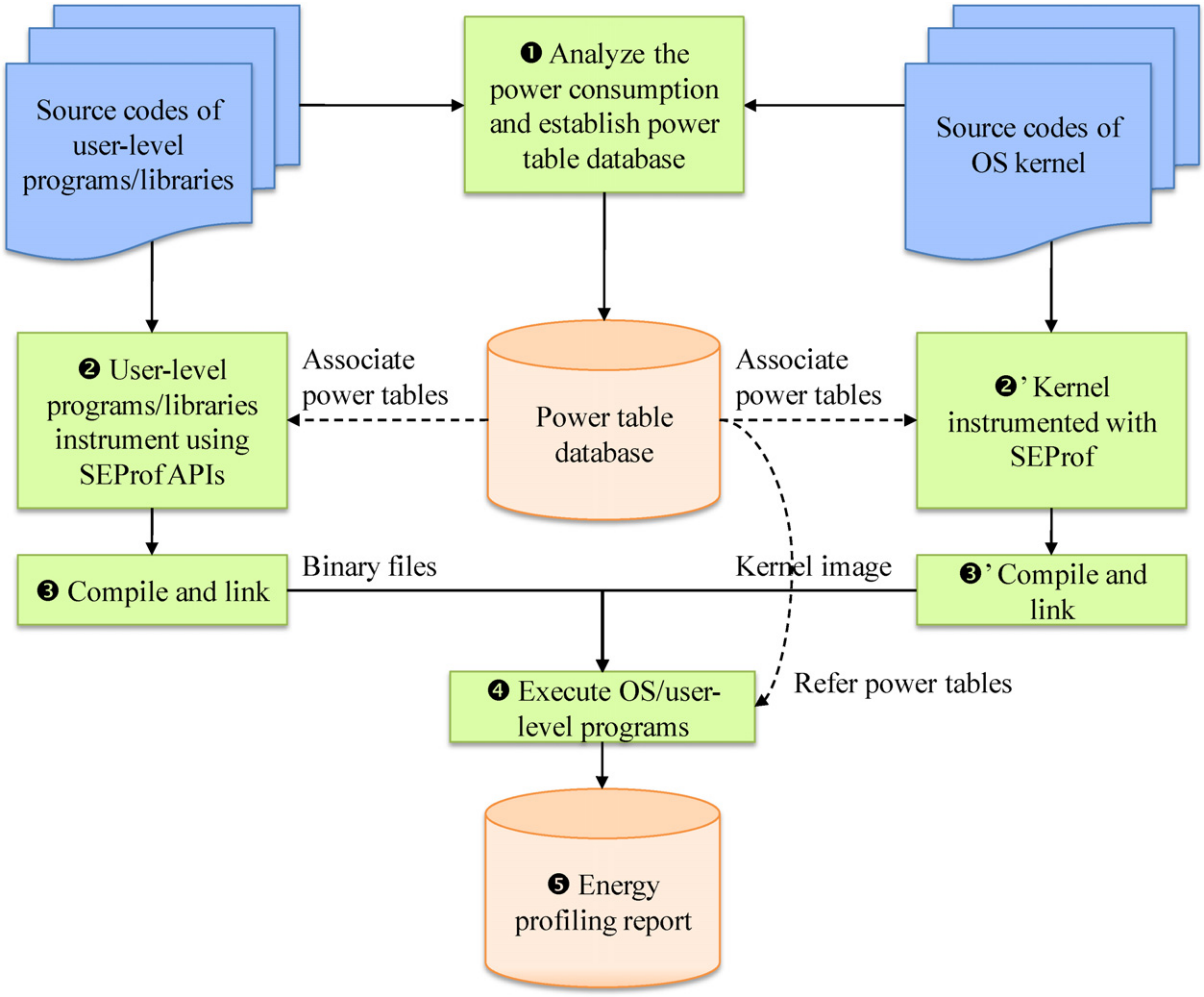

Fig. 1. Overview of SEProf and its operating steps.

tradeoff between profiling accuracy and overhead, and choose different profiling granularities in different processes and/or functions on an embedded system depending on the target accuracies. The proposed tool enables system designers to obtain the evaluation results, with sufficient profiling accuracies at a minimized profiling cost.

Since the goal of SEProf is to estimate the energy consumption of a processor enabling power management functions, a power table can consist of more than one power consumption value. Each value represents the average power consumption of multiple processes, a process, or a code block in a process executed under a specific CPU power configuration. A *CPU power configuration* represents a combination of a specific operating power mode, voltage, and frequency of the processor. Once the embedded OS activates the dynamic power management functions, it changes CPU power configuration, and SEProf ensures that the proper power consumption value is used.

After establishing the power table database in Step 1, SEProf inserts codes to user-level embedded software in Step 2 according to the desired profiling granularity. For example, a system designer can build power tables for the major functions of primary embedded software based on either a measurement-based tool or a low-level model tool such as EMSIM (Tan et al., 2002, 2003). If the processor supports five different power configurations, a power table of a function contains five power consumption values, each representing the average power for executing the function under a particular power configuration. SEProf then inserts codes in the user-level software to associate the power tables before entering the corresponding functions, and to disassociate them when leaving these functions. The instrumented codes tell SEProf which power table is associated with the running function. The instrumentation in the OS kernel is similar to user-level software, and the OS kernel functions must associate kernel power tables. Therefore, Step 2′ in Fig. 1 shows that the OS kernel is instrumented. After Step 2 and Step 2′, Step 3 and Step 3′ compile the OS kernel and the user-level programs. In Step 4, SEProf runs the program on the target embedded system, and stores the estimated energy consumption results in the kernel space. Users can access the results through SEProf application programming interfaces (APIs) in Step 5.

2.1. Power table association and power configuration setting

User-level programs and OS kernel can associate and disassociate power tables through SEProf APIs upon entering and leaving a process or code blocks of a process. The association and disassociation operations must be coupled. If a power table, say power table A, is associated, power table A will be used to estimate the average power consumption of the executing software. If another power table, say power table B, is associated before power table A is disassociated, the new power table, i.e. power table B, will be used. If power table B is disassociated, the previous power table, i.e. power table A, is then used again to estimate the power consumption of the executing software. Fig. 2 shows an example of using power tables

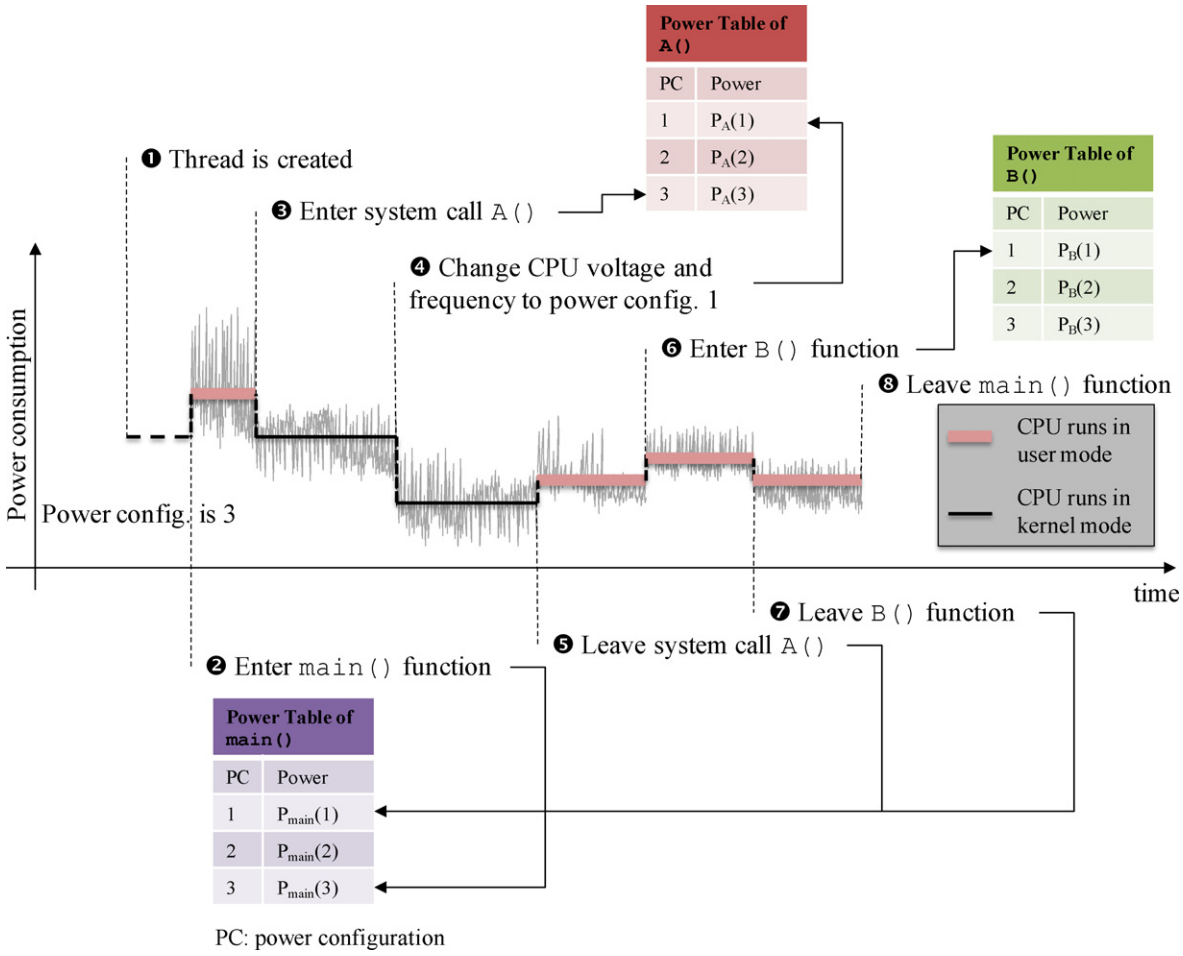

Fig. 2. An example of using power tables and power configurations in SEProf.

in SEProf. Assume that function-level power tables have been built, and the embedded processor supports three different CPU power configurations. For the ease of illustration, this example omits several detailed procedures before entering the main() function of the program. In Step 1, the CPU operates at the maximum speed in power configuration 3, and executes kernel codes to fork a thread, called *T*. SEProf initiates internal data structures for energy profiling at this stage, and associates the power tables of thread *T*'s parent with thread *T*. In Step 2, thread *T* enters its main function, and SEProf associates thread *T* with the power table of `main()`, $P_{main}(3)$. $P_{main}(3)$ denotes the CPU power consumption of running the `main()` function under CPU power configuration 3. This power table is used to estimate the power consumption in the following operations until thread *T* invokes a system call, say `A()`, in Step 3. SEProf then associates the power table of `A()` and refers to the power table of `A()`. If the OS detects the CPU is underutilized in Step 4, the OS changes the CPU power configuration from 3 to 1 to reduce energy consumption. SEProf detects this event, and changes the referred power consumption value from $P_A(3)$ to $P_A(1)$ at this stage. In Step 5, thread *T* finishes the system call, and returns to the user space. The power table of `A()` is disassociated so that the power table of `main()` is used again. In Step 6, thread *T* enters a user-level function, say `B()`. As in Step 3, the power table of `B()` and the CPU power configuration 1, i.e. $P_B(1)$, is used. In Step 7 and Step 8, the thread leaves the functions `B()` and `main()`, respectively. When thread *T* is terminated, SEProf keeps its energy consumption profile in the kernel space, and users can access the results through SEProf APIs.

2.2. Energy estimation

Fig. 3 presents a flowchart of the proposed SEProf in calculating the energy consumption of a thread. When a thread, named *T*, is created, SEProf initiates the energy profiling data structures for the thread shown as Event 1 in Fig. 3 occurs. The initialization procedure resets E_T, the accumulated energy consumption of thread *T*. The power configuration of the processor used by thread *T*, PC_T, is set to the current power configuration of the processor, PC_{cur}. The timer for measuring the execution time (which has not been used to estimate the energy consumption of thread *T*), $Timer_T$, is set to zero and paused. The power table stack of thread *T*, $PTStack_T$, which holds all associated power tables of thread *T*, is copied from that of thread *T*'s parent, and the latest associated power table of thread *T*, PT_T, is pushed to the stack $PTStack_T$. When thread *T* is scheduled (Event 2), SEProf checks whether the CPU power configuration has been modified or not by comparing the thread *T*'s associated power configuration PC_T with the current one PC_{cur}. If the power configuration is the same, SEProf resumes $Timer_T$ to measure the execution time of thread *T*. However, if the OS or other threads change the power configuration, SEProf accumulates the energy consumption of thread *T* during the execution period measured by $Timer_T$ using $E_T = E_T + Timer_T \times PT_T[PC_T]$, where $PT_T[PC_T]$ looks up the average power consumption of the processor operating at power configuration PC_T in power table PT_T. After accumulating the energy consumption, SEProf updates the power configuration PC_T to the current one PC_{cur}, and resets $Timer_T$ to accumulate the next execution period of thread *T*.

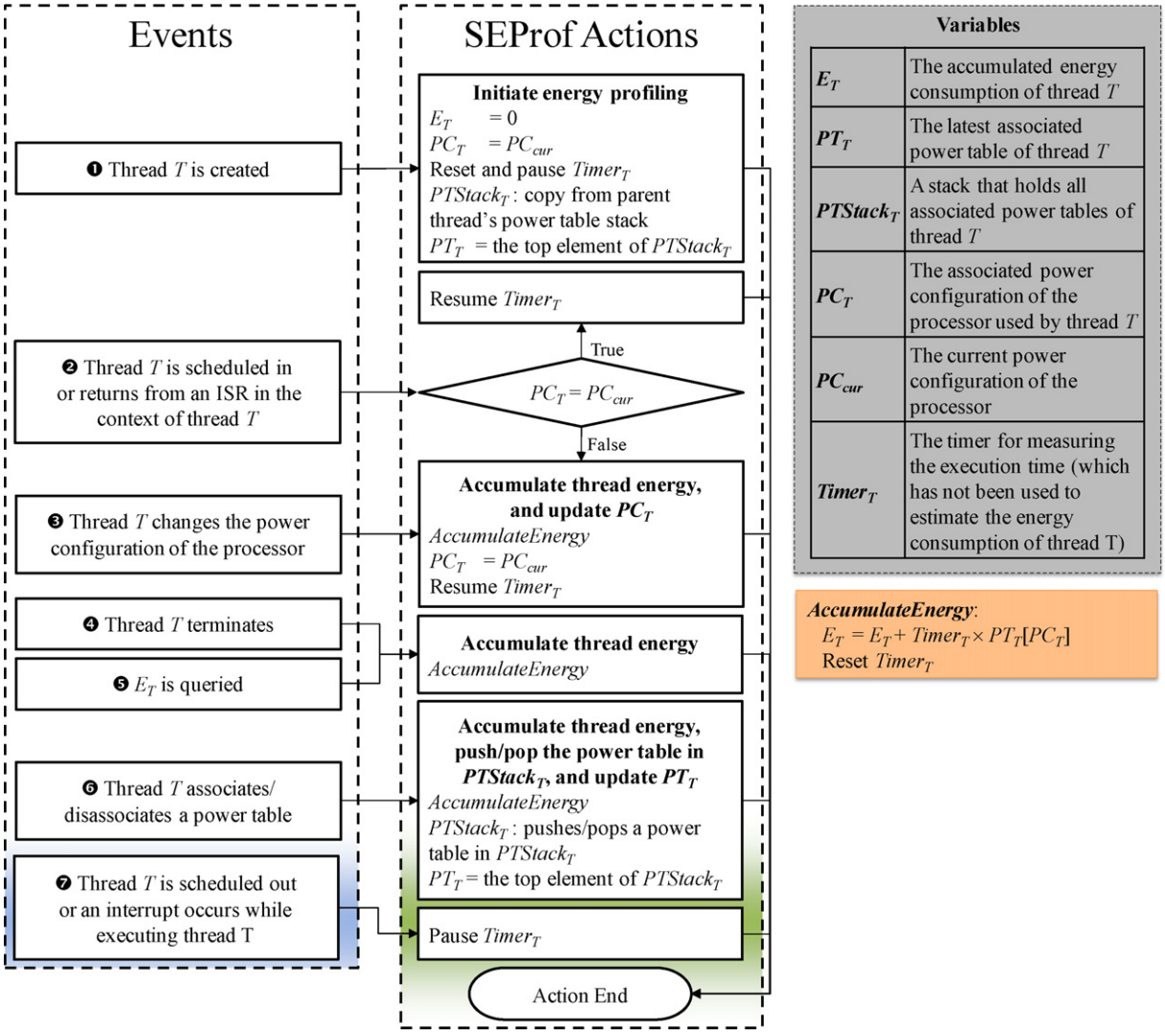

Fig. 3. Energy estimation flowchart of SEProf.

Four other events also trigger SEProf to accumulate the energy consumption of thread *T*. The first one is when thread *T* changes the CPU power configuration, PC_{cur}, indicated by Event 3. The second one is when thread *T* terminates (Event 4). The third one is when E_T is queried by thread *T* or other threads (Event 5), and the last one is when thread *T* associates or disassociates a power table (Event 6). If thread *T* associates a new power table in Event 6, the new power table becomes PT_T, and it is pushed into $PTStack_T$ after performing the energy estimation procedure. Conversely, if thread *T* disassociates a power table in Event 6, the disassociating power table is used to estimate the energy consumption, and then popped up from $PTStack_T$. The power table that appears on the top of the stack after removing the dissociating one becomes PT_T. When thread *T* is scheduled out, as shown in Event 7, SEProf pauses $Timer_T$ to stop counting the execution time of thread *T*.

In summary, SEProf accumulates the energy consumption of a thread when one of the following four events occurs.

(1) A thread associates or disassociates a power table. When a thread associates or disassociates a power table, it implies a change in the reference average power consumption of the embedded processor. Therefore, SEProf must calculate the energy consumption of the accumulated execution time and update the power table.

(2) The power configuration of the embedded processor is changed. When the CPU power configuration of an embedded processor changes, the power consumption of the processor also changes. Hence, SEProf must calculate the energy consumption of the accumulated execution time using the associated power configuration of the thread.

(3) The total energy consumption of a thread is queried. If a user queries the total energy consumption of a thread, the energy consumption of the thread must be updated before returning the energy profiling results to the user.

(4) A thread ends. When a thread terminates, the energy consumption of the thread during the last execution period is added to the total energy consumption of the thread. This is the last time that SEProf accumulates the energy consumption of the thread.

Since the OS may execute ISRs that are not a part of thread *T* when thread *T* is scheduled, SEProf could separate the energy consumption of the thread and that of ISRs by pausing $Timer_T$ when an interrupt occurs (Event 7), and resuming the timer when CPU returns from an ISR (Event 2). However, the experiments in this study did not separate these events because the runtime of ISRs is negligible.

Fig. 4 shows an example of energy estimation using *SEProf*. When a thread, say *T*, is created, the data structures of the thread

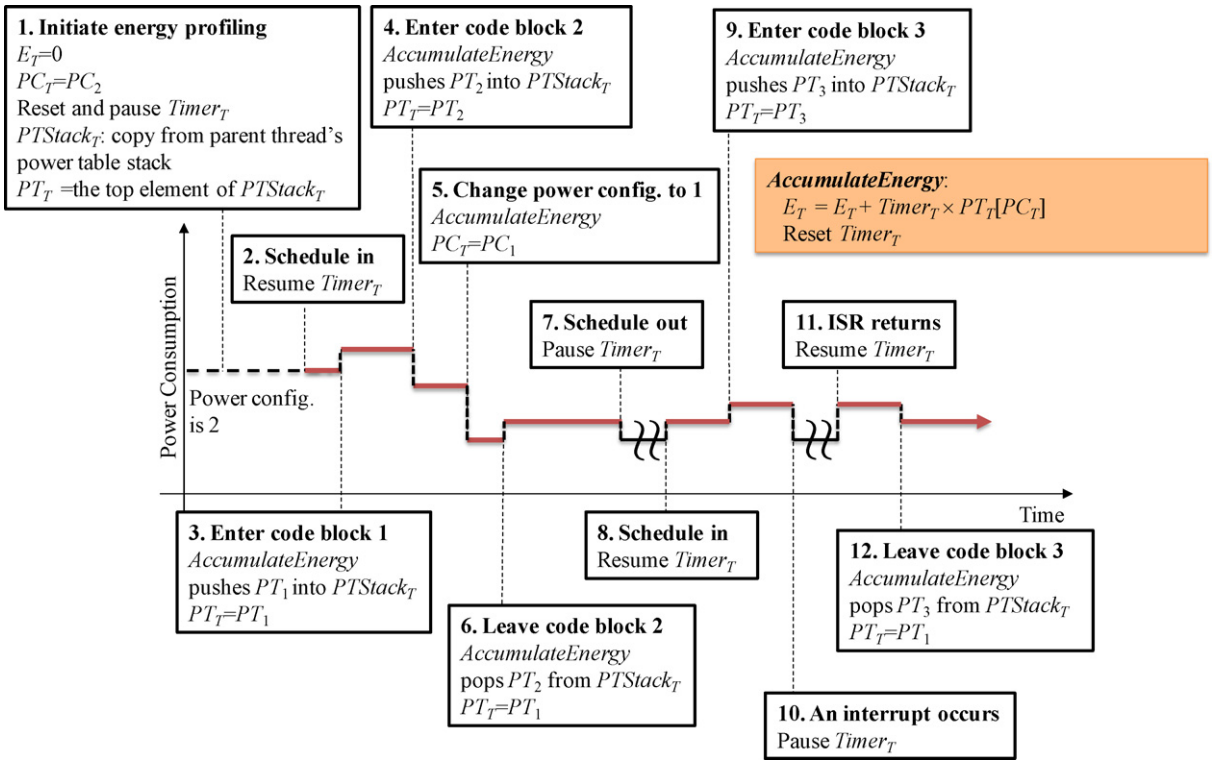

Fig. 4. An example of energy estimation using SEProf.

energy profiling are initiated in Step 1. In Step 2, thread *T* is scheduled and $Timer_T$ starts to accumulate the execution time of the thread. In Step 3, thread *T* enters code block 1 and associates with the power table PT_1. Since the power table has changed, SEProf calculates the energy consumption of thread *T* during the period from Step 2 to Step 3, resets $Timer_T$, pushes PT_1 into $PTStack_T$, and sets PT_T to PT_1. In Step 4, thread *T* enters code block 2, and associates with the power table PT_2. SEProf accumulates the energy consumption during the period from Step 3 to Step 4, pushes PT_2 into $PTStack_T$, and sets PT_T to PT_2. In Step 5, thread *T* changes the CPU power configuration from 2 to 1. SEProf accumulates the energy consumption of thread *T*, and sets PC_T to PC_1. In Step 6, thread *T* leaves code block 2, and returns to code block 1. SEProf accumulates the energy consumption during the period from Step 5 to Step 6, pops PT_2 from $PTStack_T$, and sets PT_T to the power table of code block 1. In Step 7, thread *T* is scheduled out so that the runtime measurement is paused. In Step 8, thread *T* is scheduled again. SEProf resumes $Timer_T$. In Step 9, thread *T* enters code block 3. SEProf accumulates the energy consumption during the period from Step 6 to Step 7 and Step 8 to Step 9. Afterward, SEProf pushes PT_3 into $PTStack_T$, and sets PT_T to PT_3. Because an interrupt occurs in Step 10, the execution time measurement is paused until returning from the ISR in Step 11. Finally, in Step 12, thread *T* completes the execution of code block 3, and returns to code block 1. SEProf accumulates the energy consumption during the period from Step 9 to Step 10 and Step 11 to Step 12.

2.3. Data structures

SEProf maintains three primary data structures in kernel space to support thread-based energy estimation of embedded processors enabling power management functions, as Fig. 5 shows. The following section describes these data structures:

(1) User-level program power table database. A user-level program power table database consists of the power tables of the program that are used by all threads running the same program/library. All user-level power table databases are copied into the kernel space when a thread starts running, and are accessed through indexes.
(2) A kernel power table database. A kernel power table database contains all kernel-level power tables. It is built in the OS kernel and shared among all threads on the system.
(3) A per-thread SEProf data structure in the kernel space. This data structure holds the latest associated power table, a timer variable, a power table stack, an associated power configuration, and an accumulated energy consumption of the thread.

Fig. 5 shows how SEProf maintains the power table stack. When a thread enters a new code block where a power table is associated, it pushes the associated power table into its power table stack. On the other hand, when the thread leaves the code block where the power table is disassociated, the power table is popped out from the stack. Fig. 5 shows that when the thread enters code block *i*, *j*, and *k*, the power tables of the three code blocks are linked in the power table stack.

3. Experimental results and case study

To verify the design of SEPorf and evaluate its accuracy, SEProf was implemented in Linux on an ARM11 MPCore processor (ARM, 2008). The kernel patches of Linux 2.6.19 for ARM11 MPCore platform, the source code of SEProf, and the benchmark programs used in this paper are all available on the SEProf website (SEProf, in press). Sections 3.1 and 3.2 describe the experimental environment and results, and Section 3.3 presents a cast study of using SEProf in adjusting power management strategies.

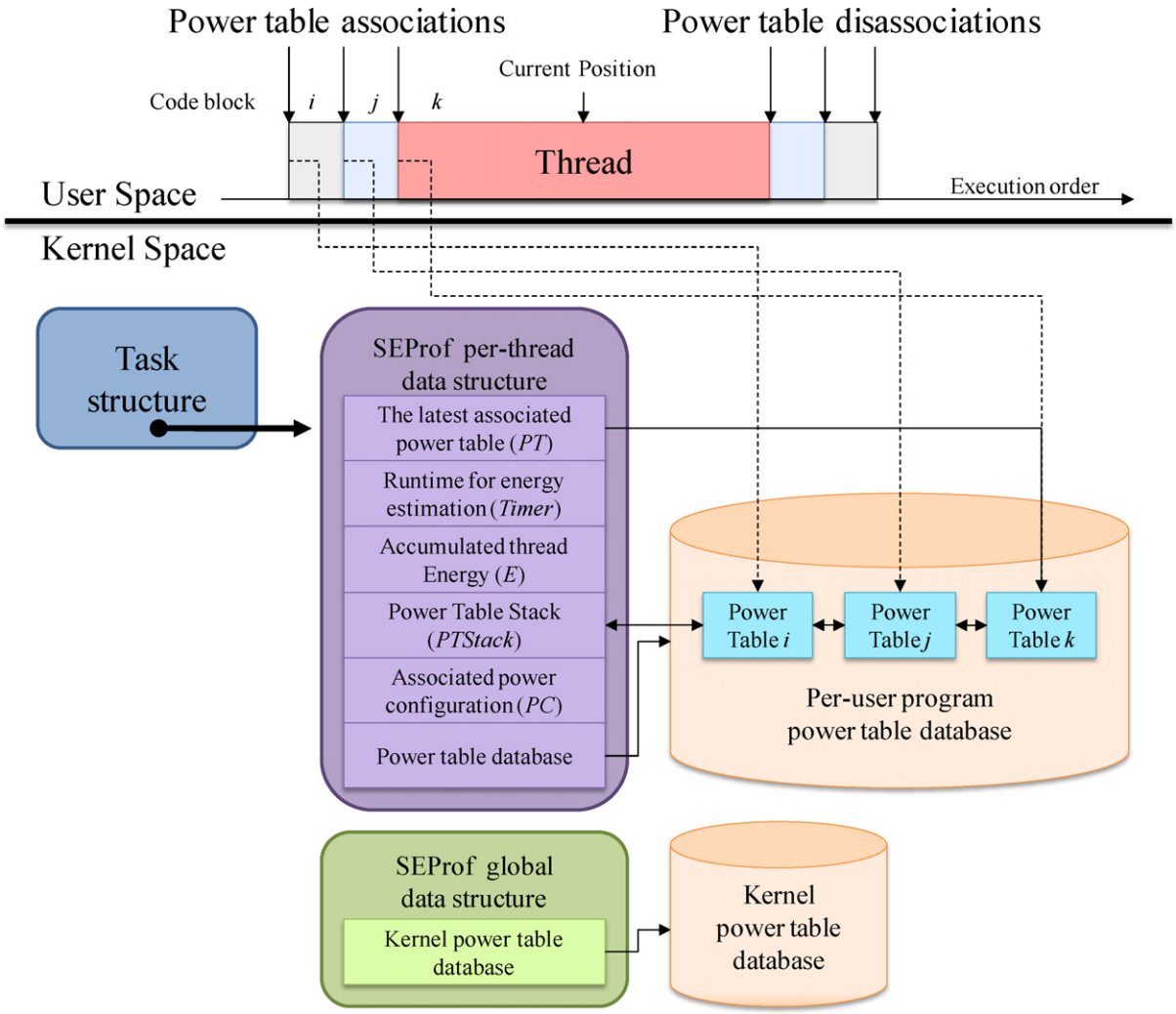

Fig. 5. Data structures maintained by SEProf.

3.1. Experimental environment

The experimental platform was a Core Tile, CT11MPCore (Core Tile, 2006), with an ARM11 MPCore test chip stacked on the top of a RealView Emulation Baseboard (Real View, 2007). This platform provided both voltage and frequency scaling functions and hardware support for measuring the voltage and current consumed by the processor. This platform made it possible to build a power table database and verify the estimation error easily. The voltage level of the ARM11 MPCore processor could be changed by writing values to the registers of an on-board digital to analog converter (DAC). ARM11 MPCore also has an on-board analog-to-digital converter (ADC) to monitor the real-time voltage and current consumption of the processor core with extremely high precision (micro volts and micro amps). The ADC stores the voltage and current consumption information on two on-board registers. The default voltage supplied to the ARM11 MPCore processor was 1.2 V, with an adjustment range of ±0.25 V. The clock rate of the processor could also be changed by configuring the phase-locked loop (PLL) on the CT11MPCore. In these experiments, the DAC and PLL were used to scale the voltage and the frequency of the ARM11 MPCore processor, respectively, and the ADC was used to measure the processor's power consumption. A 24 MHz clock on the Emulation Baseboard was used for time measurement. The time resolution is 41.7 ns.

SEProf was integrated into Linux kernel 2.6.19 with a modified OProfile (Levon, 2003) to build power table databases. OProfile is a system-wide profiler for Linux system that uses statistical sampling. OProfile originally samples the context and program counter (PC) of the running task on each sampling interrupt. We modified OProfile to gather the information of the on-board ADC registers, and can derive the power consumption of a processor. The modified OProfile can be regarded as a direct measurement tool and provides the exact power consumption of the embedded processor. Therefore, we used the OProfile results as the baseline and evaluated the accuracy of SEProf. In the experiments, the sampling rate of OProfile was set to 1 kHz.

Four benchmark programs were used throughout the experiment. The first three were CG, FT, and IS applications from the OpenMP Implementation of NAS Parallel Benchmarks (NPB) (Version 3.3) (Jin et al., 1999). CG computes an approximation to the smallest eigenvalue of a matrix using a conjugate gradient method. FT performs the time integration of a three-dimensional partial differential equation using the Fast Fourier Transform. IS sorts integers using the bucket sort. The last benchmark program, FileRW, is an I/O intensive application written by the authors. It is a simple application that writes and reads a 30 MB file through a network file system (NFS).

Table 1
Power configurations of an ARM11 MPCore processor used in the VFS experiment.

Power configuration	Voltage (V)	Frequency (MHz)
1	0.95	140
2	1.01	168
3	1.08	196
4	1.14	224
5	1.2	252

Table 2
Runtime breakdown and the average power of the benchmark programs.

Benchmark program	Runtime breakdown (%)		Average power (mW)	
	User space	Kernel space	User space	Kernel space
cg.W	99.98%	0.02%	432	425
ft.W	99.95%	0.05%	452	426
is.W	99.94%	0.06%	433	428
FileRW	0.34%	99.66%	424	409

3.2. Experimental results

This study includes two separate experiments because the ARM11 MPCore cannot dynamically change the frequency of the processor. The first one is a voltage and frequency scaling (VFS) experiment, and the second one is a dynamic voltage scaling (DVS) experiment. In the VFS experiment, both the voltage and the frequency of the ARM11 MPCore processor were scaled at the beginning of the experiment and remained the same throughout the experiment. In the DVS experiment, the voltage of the ARM11 MPCore processor was scaled dynamically and periodically.

3.2.1. VFS experiment

The VFS experiment selected five power configurations for the ARM11 MPCore processor, and configured the processor to operate under one of five power configurations during the experiment. As Table 1 shows, each power configuration represents a combination of voltage and frequency levels for the processor. Throughout the experiment, only one ARM11 CPU was active to map the measured power consumption back to the embedded software. The remaining three CPUs were not initialized. Tables 2 and 3 provide information regarding the structure and characteristics of the benchmark programs. Table 2 shows the runtime breakdown and the average power dissipation for the execution of each benchmark program under power configuration 3. Programs cg.W, ft.W, and is.W consumed over 99% of runtime in the user space, while FileRW consumed 99% of the runtime in the kernel space. Table 3 further provides the runtime breakdown of the major functions for the four programs. The major functions involved in FileRW are all kernel internal functions.

Table 3
Runtime breakdown and the average power of the major functions (more than 2% of runtime).

Benchmark program	Function name	% of runtime	Average power (mW)
cg.W	conj_grad	95.34%	431
	sparse	3.04%	459
ft.W	fftz2	43.88%	452
	cffts1	9.57%	455
	cffts2	9.46%	455
	cffts3	9.37%	451
is.W	rank	73.52%	433
	randlc	17.13%	434
FileRW	smsc911x_poll	56.89%	408
	__copy_to_user	10.70%	410
	__copy_from_user	6.01%	418
	__memzero	3.51%	409
	csum_partial_copy_nocheck	3.43%	406
	smsc911x_hard_start_xmit	2.40%	408

The results in Tables 2 and 3 indicate that the major functions of a benchmark program may consume similar power, but the power consumption of different programs is quite different. Therefore, in the VFS experiment, seven power tables were built for the four benchmark applications, busybox, the modified OProfile, and the Linux kernel, as shown in Table 4. Each power table consisted of five entries under five different CPU power configurations. The power tables were built by executing each benchmark program for more than ten seconds under a specific power configuration on Linux kernel. The modified OProfile ran simultaneously to measure and collect the power consumption of the embedded processor. Once the benchmark program was executed under the five different power configurations of the embedded processor, the power tables of the benchmark program were constructed. The power table of each application was associated with SEProf at the beginning of the application, and disassociated at the end. All applications shared the same kernel power table, vmlinux. The kernel power table is associated when a thread calls a system call, and disassociated when the thread returns from the system call. It is also associated with threads that have no dedicated power tables.

Table 5 depicts the energy estimation results of the benchmark programs by using SEProf. The energy and time spent on executing application itself and calling system calls were separated to better examine the accuracy of the power estimation results. Table 5 shows that, in many cases, the average power consumption of the application was slightly lower than that of the average power consumption in Table 4. This is because the parent threads of the benchmark programs have no dedicated power tables, so they associate with the kernel power table, vmlinux, which has the lowest average power. When the benchmark programs were executed, they used the kernel power table copied from the parent threads until they associated with their own power tables.

Table 6 verifies the accuracy of the power estimation results for VFS experiments. OProfile was modified to sample the measured power of the ARM11 MPCore and compare it with the estimated power provided by SEProf. Table 6 shows the mean absolute power estimation error of the four benchmark programs and an overall period. The overall period began when the `init` process executed command scripts for system startup, and ended when all benchmark programs terminated. It represents the execution of Linux kernel and applications including the benchmark programs and the other programs without dedicated power databases. These results show that the power estimation error using SEProf was quite low. In most cases, the average estimation error was less than 2% and the standard deviation of the estimation error was less than 2%.

In Table 7, we compared the execution time of benchmark programs on a non-instrumented system and an instrumented system with SEProf. Each benchmark program was executed for 10 times. To eliminate cache miss effects in the first several runs, we calculated the execution time of a benchmark program based on the last seven runs. The execution timer started when a benchmark program was launched and stopped once the program terminated. During the entire period of a benchmark program execution, no other program was running. The performance overhead introduced by using SEProf was derived by comparing the execution time when a benchmark program ran on an instrumented and a non-instrumented system. The average performance overhead introduced by SEProf in the VFS experiment is less than 5%.

Table 8 further illustrates the profiling accuracy and overhead when system designers specify different granularities in SEProf to profile embedded software. The number of power table operations in Table 8 counts the number that a benchmark program associates or disassociates a power table. In the experiment, a

Table 4
Power tables used in the VFS experiment.

Power configuration	Average power (mW)						
	busybox	cg.W	ft.W	is.W	FileRW	OProfiled	vmlinux
1	260	249	258	246	246	264	236
2	352	335	349	334	330	359	320
3	460	438	458	438	431	470	419
4	589	559	586	560	565	602	537
5	738	696	731	701	692	753	668

Table 5
Energy estimation results of the benchmark programs generated by SEProf.

Power configuration	Benchmark program	Average power (mW)	Energy breakdown	
			Application Average power (mW)	System call/kernel Average power (mW)
1	cg.W	249	249	236
	ft.W	258	258	236
	is.W	246	246	236
	FileRW	236	237	236
2	cg.W	335	335	320
	ft.W	349	349	320
	is.W	334	334	320
	FileRW	320	321	320
3	cg.W	438	438	419
	ft.W	458	458	419
	is.W	438	438	419
	FileRW	419	421	419
4	cg.W	559	559	537
	ft.W	586	586	537
	is.W	560	560	537
	FileRW	537	540	537
5	cg.W	696	696	668
	ft.W	731	731	668
	is.W	701	701	668
	FileRW	668	672	668

Table 6
Power estimation error in the VFS experiment.

Power configuration	Benchmark program/overall	Number of samples (1 ms/sample)	Average absolute estimation error	Standard deviation
1	cg.W	227,437	0.05%	1.16%
	ft.W	70,929	0.05%	1.39%
	is.W	34,767	0.14%	0.76%
	FileRW	16,935	0.97%	1.52%
	Overall	392,000	0.04%	1.42%
2	cg.W	203,952	0.30%	1.20%
	ft.W	61,413	0.16%	1.49%
	is.W	30,003	0.22%	0.74%
	FileRW	15,836	0.97%	1.53%
	Overall	346,787	0.20%	1.49%
3	cg.W	188,774	0.34%	1.16%
	ft.W	54,521	0.20%	1.59%
	is.W	26,756	0.36%	0.69%
	FileRW	14,838	0.95%	1.48%
	Overall	315,333	0.29%	1.52%
4	cg.W	176,394	0.18%	1.10%
	ft.W	49,043	0.15%	1.70%
	is.W	24,308	0.27%	0.61%
	FileRW	14,004	1.05%	1.28%
	Overall	291,048	0.10%	1.54%
5	cg.W	167,235	0.12%	0.99%
	ft.W	43,609	0.15%	1.62%
	is.W	22,380	0.27%	0.67%
	FileRW	13,467	0.71%	1.19%
	Overall	272,056	0.13%	1.50%

Table 7
Performance overhead of using SEProf in the VFS experiment.

Power configuration	Benchmark program	% of SEProf overhead
1	cg.W	0.56%
	ft.W	3.73%
	is.W	0.88%
	FileRW	2.55%
2	cg.W	0.99%
	ft.W	4.30%
	is.W	0.24%
	FileRW	1.25%
3	cg.W	0.19%
	ft.W	4.66%
	is.W	0.27%
	FileRW	2.71%
4	cg.W	0.10%
	ft.W	4.29%
	is.W	0.74%
	FileRW	1.67%
5	cg.W	0.09%
	ft.W	0.38%
	is.W	1.21%
	FileRW	0.79%

single per-system power table was used for all processes on the embedded system. The profiling accuracy was calculated by comparing the estimated results and the real power consumption of the system. The profiling overhead was measured by checking the performance differences between an instrumented and non-instrumented system. For a fair comparison, the extra power consumption introduced by the measurement tool and SEProf was removed from the measurement and estimation results. The experimental results indicate that fine-grain power tables generally achieve better accuracy, but introduce more profiling overhead than coarse-grain power tables.

3.2.2. DVS experiment

As in the VFS experiment, the DVS experiment selected five power configurations for the ARM11 MPCore processor, and activated only one ARM11 CPU. However, the clock frequency of the processor operating at each power configuration was the same as in Table 9, and the voltage of the processor varied at runtime. Seven power tables were built for the applications and the Linux kernel, respectively, as Table 10 shows.

In the DVS experiment, the voltage of the processor was periodically scaled at three different time intervals: 100 ms, 1 s, and 10 s. The power configuration of the processor was increased by one at each time interval. If the power configuration of the processor reached five, it was set to one in the next time interval. Fig. 6 shows an example of DVS and its power consumption. In this example, two lines show the measured and the estimated power consumption during the execution of the IS application. Because the DVS interval was set to 100 ms in this example, the power consumption of the processor varied every 100 ms. Fig. 6 shows that the estimated power consumption was very close to the measured one. The estimated power consumption occasionally drops, but the measured one does not. This is because that the thread that executed the IS application was scheduled out during that period, and another thread which associated with a different power table was scheduled in. If the newly scheduled thread had a lower average power consumption, then a drop appears in the figure. Since the measured power consumption read from the ADC is updated every 5 ms, the drop of the real power consumption cannot be detected if the newly scheduled thread is scheduled out within the ADC update period.

Table 9
Power configurations of an ARM11 MPCore processor used in the DVS experiment.

Power configuration	Voltage (V)	Frequency (MHz)
1	0.95	140
2	1.01	140
3	1.08	140
4	1.14	140
5	1.2	140

Table 11 presents the power estimation error in the DVS experiment. The average absolute estimation error and the standard deviation of the error increased as the DVS interval decreased. This is because the power consumption of the processor does not change immediately after a new value is written to the DAC, and the power consumption of the processor after changing voltages is unable to be read from the ADC immediately. Fig. 7 illustrates this using

Table 10
Power tables used in the DVS experiment.

Power configuration	Average power (uW)						
	busybox	cg.W	ft.W	is.W	FileRW	OProfiled	vmlinux
1	260	249	258	246	246	264	236
2	298	284	295	282	282	302	271
3	339	322	336	321	330	344	309
4	382	362	378	363	370	388	349
5	427	403	422	406	408	432	389

Table 8
Profiling accuracy and overhead when using different number of power tables.

Number of power tables	Benchmark program	% of SEProf overhead	Number of power table operations		Average absolute estimation error	Standard deviation
			User space (system call)	Kernel space		
One per-system power table	cg.W	0.22%	0	256	0.95%	1.07%
	ft.W	0.56%	0	252	3.24%	1.79%
	is.W	0.19%	0	206	0.91%	0.81%
	FileRW	4.36%	0	136	6.70%	1.56%
7 per-process power tables	cg.W	0.19%	2	256	0.04%	1.39%
	ft.W	4.66%	2	252	0.09%	1.60%
	is.W	0.27%	2	206	0.09%	1.02%
	FileRW	2.71%	2	136	0.00%	1.62%
24 power tables for major functions of processes	cg.W	0.32%	36	4447	0.06%	0.97%
	ft.W	7.75%	36	4447	0.07%	1.57%
	is.W	517.20%	8,388,632	4209	0.07%	0.98%
	FileRW	4.86%	2	218,483	0.00%	1.62%

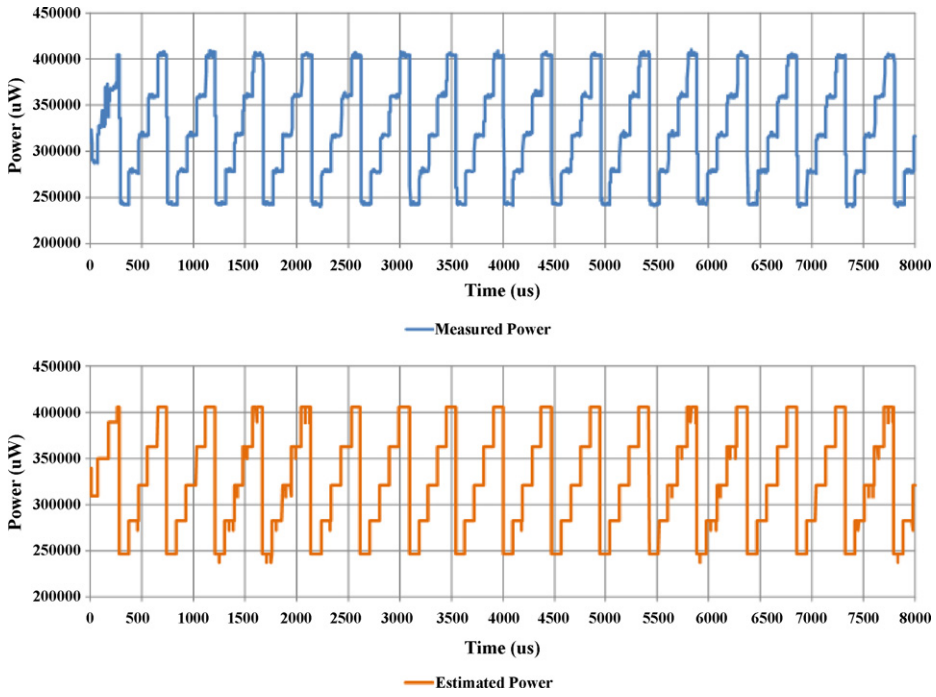

Fig. 6. The measured and the estimated power consumption during the execution of is.W.

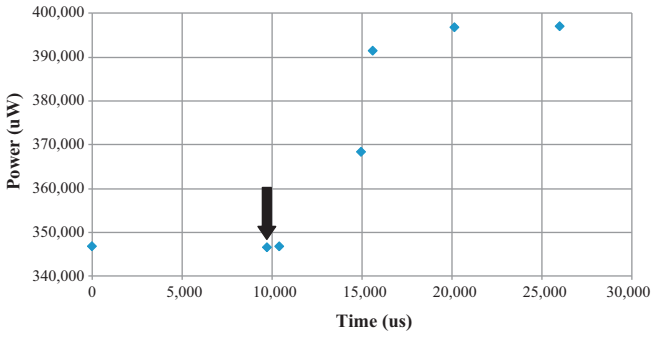

Fig. 7. Power samples during DVS.

seven power samples taken from the ADC during the period that the voltage level of the processor is scaling. The arrow in Fig. 7 indicates the time that the new voltage level is written to the DAC. SEProf updates the power configuration of the processor at this point, but the power consumption of the processor does not change immediately. Instead, it becomes stable, and can be read form ADC 10 ms later. Consequently, the power consumption difference between the measured values and the estimated values during this period increases the average estimation error and the standard deviation of the error.

In Table 12, we compared the execution time of benchmark programs on a non-instrumented system and an instrumented system with SEProf and seven power tables. Each benchmark program was executed for ten times, and the average execution time was calculated based on the last seven runs to avoid cache miss effects in the first several runs. Furthermore, the instrumented system altered the voltage periodically. The average performance overhead introduced by SEProf in the DVS experiment is also less than 5%.

3.3. A case study of using SEProf in adjusting power management strategies

With the aid of SEProf, system designers are able to obtain fast evaluation of the power consumption of an embedded system, and adjust dynamic power management strategies efficiently. For

Table 11
Power estimation error in DVS experiment.

DVS interval	Benchmark program/overall	Number of samples (1 ms/sample)	Average absolute estimation error	Standard deviation
100 ms	cg.W	227,888	1.18%	4.65%
	ft.W	71,085	0.88%	4.96%
	is.W	34,674	0.78%	5.03%
	FileRW	16,743	1.65%	4.88%
	Overall	394,869	1.08%	4.85%
1 s	cg.W	228,028	0.08%	1.94%
	ft.W	70,887	0.06%	2.08%
	is.W	34,688	0.00%	1.69%
	FileRW	17,027	0.69%	2.20%
	Overall	393,616	0.09%	2.10%
10 s	cg.W	228,118	0.26%	1.23%
	ft.W	70,943	0.10%	1.60%
	is.W	34,986	0.30%	1.01%
	FileRW	16,767	0.90%	1.20%
	Overall	393,227	0.21%	1.49%

Table 12
Performance overhead of using SEProf in the DVS experiment.

DVS interval	Benchmark program	% of SEProf overhead
100 ms	cg.W	0.28%
	ft.W	3.74%
	is.W	0.73%
	ExeFileRW	4.50%
1 s	cg.W	0.16%
	ft.W	3.93%
	is.W	0.01%
	ExeFileRW	3.36%
10 s	cg.W	0.25%
	ft.W	3.84%
	is.W	0.77%
	ExeFileRW	1.68%

Table 13
Average frame rate of the MPEG-4 decoder under different power configurations.

Power configuration		User space				
		1	2	3	4	5
Kernel space	1	24.13	27.68	31.01	34.08	36.75
	2	24.80	28.57	32.13	35.44	38.34
	3	25.19	29.08	32.78	36.23	39.27
	4	25.56	29.59	33.43	37.02	40.20
	5	25.89	30.02	33.98	37.70	41.00

example, we implemented a networked media player based on an open-source MPEG-4 decoder on the experimental platform. The networked media player periodically downloaded a video from a network node and decoded the video. Like Step 1 in Fig. 1, we first used the modified OProfile tool to established the power tables for the networked media player. In this case study, we built both process-level and function-level power tables of the networked media player. After Step 2 and Step 3 in Fig. 1, we can start to evaluate the power consumption of the embedded system. First, we used process-level power tables and performed the evaluation like Step 4 in Fig. 1. The report revealed that the power consumption of the embedded system is dominated by the networked media player. Then, we applied function-level power tables of the networked media player and performed the evaluation again. We found that the frame decoding function consumes a significant power, but the frame rate is more than 40 frames/s which is higher than our target performance. We then considered a simple power management strategy which determines the speed and power mode of CPU so that 30 frames/s performance requirement can be achieved with the minimized power consumption. We implemented the power management strategy in the networked media player, re-compiled the program, and re-ran the energy estimation again under different CPU power configurations. Based the reports generated by SEProf, we can obtain Tables 13 and 14 which show the performance in terms of the average frame rate of the MPEG-4 decoder, and the average power consumption of the embedded system under different combinations of CPU power configurations, respectively. The five CPU power configurations are the same as these shown in Table 1. As we can see from the tables, if our target performance, i.e. the average frame rate of the networked media player, is set to 30 frames/s, the best dynamic power management strategy which can minimize the power consumption of the system is to set the CPU into power configuration 1 when it runs kernel and changes the CPU to power configuration 3 when it runs the MPEG-4 decoder. The reason behind this strategy is because that the processor needs sufficient processing speeds (power configuration 3 and above) to handle video decoding and achieve the target frame rate. However, when the processor runs kernel, it usually handles I/O operations and it does not require a high processing power. Therefore, if the processor can handle sufficient I/O operations, to slow down the processor speed when the processor runs kernel is beneficial to reduce overall power consumption of the embedded system. The case study demonstrated how this tool can benefit developers to identify the energy consumption problems, speed up the evaluation process, and help developers in adjusting the power management strategies.

Table 14
Average energy consumption (μJ) for decoding a video frame under different power configurations.

Power configuration		User space				
		1	2	3	4	5
Kernel space	1	10,553	11,757	13,114	14,567	16,199
	2	10,893	12,097	13,454	14,906	16,539
	3	11,383	12,586	13,943	15,396	17,028
	4	11,858	13,062	14,419	15,871	17,504
	5	12,365	13,568	14,925	16,378	18,010

The energy consumption may not be the only criteria in developing embedded systems. Developers may have to use different profiling tools such as performance and energy tools, examine the system from different aspects, and explore solution spaces during the design phase. For example, in the case study shown in Section 3.3, the frame rate of an MPEG-4 decoder is a critical performance factor. Therefore, besides the energy estimation conducted by SEProf, we also evaluated the performance of an MPEG-4 decoder in terms of frame rate per second. By evaluating the performance of the MPEG-4 decoder, developers know that there are 16 CPU power configurations which can meet the performance requirement, i.e. decoding 30 frames/s. Without the assistance of an energy profiling tool, developers may consider the solution which can just meet the performance requirement, i.e. the 30.02 frames/s solution, to save energy. However, according to the energy profiling results shown in Table 14, the 30.02 frames/s solution consumes more energy than the 31.01 frames/s solution and the 32.13 frames/s solution. The case study reveals that to combine the performance and energy profiling analyses can facilitate developers to explore alternative solutions for efficient software designs.

4. Conclusions

In this paper, we proposed a high-level energy profiling tool called SEProf. SEProf provides fast run-time power estimation of a complex embedded system so that system designers can adjust dynamic power management strategies for the embedded software efficiently. Moreover, SEProf supports energy profiling at different granularities and enables system designers to make a tradeoff between profiling accuracy and overhead. We implemented SEProf in Linux kernel 2.6.19, and conducted a number of experiments on an ARM11 MPCore processor. The experimental results show that the average power estimation error using SEProf is within 2%, and the performance overhead introduced by SEProf is less than 5%.

Acknowledgments

The authors would like to thank Industrial Technology Research Institute, MediaTek Inc. and National Science Council of the Republic of China for financially supporting this research under Contract No. NSC100-2219-E-009-022-, NSC100-2220-E-009-038-, NSC99-2220-E-009-045-, NSC101-3113-P-006-020-, NSC101-2219-E-009 -001-, and NSC99-2915-I-009-064.

References

ARM11 MPCore Processor Revision r1p0 Technical Reference Manual, ARM, February 2008.

Blume, H., Becker, D., Rotenberg, L., Botteck, M., Brakensiek, J., Noll, T.G., 2007a. Hybrid functional- and instruction-level power modeling for embedded and heterogeneous processor architectures. Journal of Systems Architecture 53 (10), 689–702.

Blume, H., Livonius, J. v., Rotenberg, L., Noll, T.G., Bothe, H., Brakensiek, J., 2007b. Performance and power analysis of parallelized implementations on an MPCore multiprocessor platform. In: International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (IC-SAMOS).

Brooks, D., Tiwari, V., Martonosi, M., 2000. Wattch: a framework for architectural-level power analysis and optimizations. In: 27th International Symposium on Computer Architecture (ISCA-27), June 2000.

Burger, D., Austin, T.M., 1997. The SimpleScalar tool set, Version 2.0. Computer Architecture News (June), 13–25.

Choi, K., Soma, R., Pedram, M., 2005. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24 (January (1)).

Core Tile for ARM11 MPCore HBI-0146 User Guide, ARM, September 2006.

Flinn, J., Satyanarayanan, M., 1999. PowerScope: a tool for profiling the energy usage of mobile applications. In: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications (WMCSA).

Hsu, C.-H., Chen, J.-J., Tsao, S.-L., 2007. Evaluation and modeling of power consumption of a heterogeneous dual-core processor. In: 13th International Conference on Parallel and Distributed Systems (ICPADS), December 2007, Hsinchu, Taiwan.

Isci, C., Buyuktosunoglu, A., Cher, C.-Y., Bose, P., Martonosi, M., 2006. An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget. In: The 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).

Jin, H., Frumkin, M., Yan, J., 1999. The OpenMP implementation of NAS parallel benchmarks and its performance, NAS Technical Report NAS-99-011, NASA Ames Research Center, October.

Kumar, R., Farkas, K.I., Jouppi, N.P., Ranganathan, P., Tullsen, D.M., 2003. Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction. In: Proceedings of the 36th International Symposium on Microarchitecture (MICRO), December 2003.

Levon, J., 2003. OProfile Internals, http://oprofile.sourceforge.net/doc/internals/index.html.

Li, T., John, L.K., 2003. Run-time modeling and estimation of operating system power consumption. In: Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems.

Monchiero, M., Canal, R., Gonzalez, A., 2008. Power/performance/thermal design-space exploration for multicore architectures. IEEE Transactions on Parallel and Distributed Systems 19 (May (5)).

Qu, G., Kawabe, N., Usami, K., Potkonjak, M., 2000. Function-level power estimation methodology for microprocessors. In: Proceedings of Design Automation Conference (DAC), pp. 810–813.

RealView™ Emulation Baseboard HBI-0140 Rev D User Guide, ARM, October 2007.

SEProf, http://brass.cs.nctu.edu.tw/SEProf.

Sinha, A., Chandrakasan, A.P., 2001. Jouletrack—a web based tool for software energy profiling. In: Proceedings of the Design Automation Conference (DAC).

Tan, T.K., Raghunathan, A., Lakshminarayana, G., Jha, N.K., 2001. High-level software energy macro-modeling. In: Proceedings of Design Automation Conference, June 2001.

Tan, T.K., Raghunathan, A., Jha, N.K., 2002. EMSIM: an energy simulation framework for an embedded operating system. In: Proceedings of IEEE International Symposium on Circuit & Systems, May 2002, pp. 464–467.

Tan, T.K., Raghunathan, A., Jha, N.K., 2003. A simulation framework for energy consumption analysis of OS-driven embedded applications. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22 (September (9)), 1284–1294.

Tiwari, V., Malik, S., Wolfe, A., 1994. Power analysis of embedded software: a first step towards software power minimization. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2 (December (4)), 437–445.

Shiao-Li Tsao earned his PhD degree in engineering science from National Cheng Kung University, Taiwan in 1999. His research interests include embedded software and system, and mobile communication and wireless network. He was a visiting scholar at Bell Labs, Lucent technologies, USA, in the summer of 1998, a visiting professor at Dept. of Electrical and Computer Engineering, University of Waterloo, Canada, in the summer of 2007, and Dept. of Computer Science, ETH Zurich, Switzerland, in the summer of 2010 and 2011. From 1999 to 2003, Dr. Tsao joined Computers and Communications Research Labs (CCL) of Industrial Technology Research Institute (ITRI) as a researcher and a section manager. Dr. Tsao is currently an associate professor of Dept. of Computer Science of National Chiao Tung University. Prof. Tsao has published more than 75 international journal and conference papers, and has held or applied 18 US patents. Prof. Tsao received the Research Achievement Awards of ITRI in 2000 and 2004, Highly Cited Patent Award of ITRI in 2007, Outstanding Project Award of Ministry of Economic Affairs (MOEA) in 2003, and Advanced Technologies Award of MOEA in 2003. He also received the Young Engineer Award from the Chinese Institute of Electrical Engineering in 2007, Outstanding Teaching Award of National Chiao Tung University, and K. T. Li Outstanding Young Scholar Award from ACM Taipei/Taiwan chapter in 2008. He is a member of IEEE.

Jian-Jhen Chen received his master's degree in computer science from National Chiao Tung University, Taiwan in 2009. He is currently a PhD candidate in the same university. His main research interests are in the field of power estimation and energy-efficient software design for embedded systems. He received an academic achievement award from National Chiao Tung University in 2006, and a Tatung chairman award from Tatung University, Taiwan in 2006.