ePRO: A Tool for Energy and Performance Profiler for Embedded Applications[1]

Woongki Baek
School of CSE
Seoul National University
Seoul, Korea
wkb@davinci.snu.ac.kr

Young-Jin Kim
School of CSE
Seoul National University
Seoul, Korea
youngjk@davinci.snu.ac.kr

Jihong Kim
School of CSE
Seoul National University
Seoul, Korea
jihong@davinci.snu.ac.kr

Abstract – *Energy and performance are two of the most important parameters in designing embedded systems. In this paper, we describe the architecture and implementation of ePRO, a tool for energy and performance profiler for embedded applications. Energy profiling is mainly based on the hardware instrumentation which measures current samples during execution of embedded applications with the support of a periodic kernel module. Performance profiling is done with little side effect to embedded applications by using a performance monitoring unit which is provided in the architectural level. Using ePRO, we could improve the energy consumption and the performance of a digital signal processing application by 5.4% and 4.4%, respectively.*

Keywords: energy profiling, performance optimization, low-power, embedded software

1 Introduction

For designing efficient embedded applications, developers should consider two constraints. First, the performance of an embedded application should be maximized on a given embedded system. Second, the energy consumption should be minimized as much as possible because energy is usually a very critical resource in embedded systems. These are mainly due to the limited CPU performance and the limited battery capacity of embedded systems. The performance and the energy consumption of embedded applications can not be considered totally distinct. For example, if a software implementation of MPEG4 decoder has a very strong performance but also requires a large amount of energy consumption, it is not suitable for embedded systems. If another software implementation of MPEG4 is very efficient in energy consumption, but has a very poor performance, it can not be used neither. Therfore, considering both performance and energy consumption is very important in designing efficient applications.

To help developers of embedded applications to design embedded software with high-performance and low-power consumption, the demand for a tool which can map the profiled information of energy consumption and performance to program structure is increasing. In this paper, we describe the architecture and implementation of energy and performance profiler (ePRO).

ePRO attributes profiles of performance and energy consumption to program structure. For measuring the energy consumption of embedded applications, we used the similar approach proposed by Flinn et al. [1], which combines hardware instrumentation with kernel software support. To profile performance information, we assumed the specific architecture, XScale [2]. By converting the source code of embedded applications to use a performance monitoring unit (PMU) in XScale, we can collect various performance data, such as instruction cache efficiency, data cache efficiency, instruction fetch latency, data/bus request bus full, stall/writeback statistics, instruction TLB efficiency and data TLB efficiency with little overhead.

The contribution of this paper is two-fold. First, to the best of our knowledge, ePRO is the first tool which profiles both performance and energy consumption data on a real-world embedded system. Second, we present a case study on a digital signal processing (DSP) application as a guideline for using our tool. In the case study, we obtained 5.4% of reduction in total energy consumption while improving performance by 4.4% using ePRO.

The rest of this paper is organized as follows. Sections 2 discusses related work. Section 3 describes design requirements of ePRO. Section 4 presents the architecture and implementation of ePRO. Section 5 discusses a case study for a DSP application. Finally, section 6 concludes.

2 Related work

Existing energy estimation and monitoring techniques can be divided into two categories: simulation-based or measurement-based. Energy simulators such as Wattch [3] and SimplePower [4] estimate the energy consumption in reasonable time. However, their accuracy is not so high that energy optimization is difficult by using those simulators.

[1] This work was supported by Electronics and Telecommunications Research Institute under contract # 1010-2004-0098

Representative researches in measure-based estimation techniques are SES [5] and PowerScope [1]. SES is an energy monitoring tool which collects energy consumption data in a cycle-by-cycle resolution and maps the collected energy consumption data to program structure. SES has a main advantage that the accuracy of analysis results is very high because profiling is performed in a cycle-by-cycle resolution. However, SES needs an extra profile acquisition module which consists of measurement circuit, profile controller and acquisition memory. Therefore, the techniques used in SES may not be applied to ordinary embedded systems which do not equip with profile acquisition modules.

PowerScope [1] is based on hardware instrumentation by using a digital multimeter with support of embedded operating system. Profiled energy costs are mapped to the procedure level of high-level language. In PowerScope, Any extra hardware logic is not needed in embedded systems. Therefore, PowerScope is applicable to ordinary embedded systems.

ePRO employs measure-based estimation techniques used in SES and PowerScope. However, ePRO is distinct from SES because ePRO does not need any extra hardware module such as profile acquisition module in SES. Therefore, ePRO can be used on ordinary embedded systems. The main advantage of ePRO over PowerScope lies in performance profiling. ePRO has a performance profiling module while PowerScope only profiles energy consumption. By profiling both energy consumption and performance statistics, we believe ePRO will be more helpful in developing efficient embedded applications. Moreover, unlike PowerScope, the daemon of the system-monitoring module was removed and the function call path for performance profiling was simplified to reduce the overhead.

3 Design requirements

There exist several requirements in designing energy and performance profiling tool. First, the tool should profile energy consumption and performance of embedded applications as accurately as much possible. The tool should have high resolution in sampling energy and performance data to map the profiled information to specific areas in embedded applications. If profiling is performed in a very low sampling rate, the analysis of energy consumption and performance of embedded applications will not be trustable.

Second, the tool should profile energy consumption and performance with little overhead. If the overhead incurred by the tool is too high, profiled energy consumption and performance data will be less meaningful, because the behavior of an embedded application is changed so much by the overhead of the tool.

Finally, the profiled energy consumption and performance data should be shown in a user-friendly format. The tool should help programmers to identify hot-spots of their embedded applications easily and to optimize these hot-spots in perspective of energy and performance.

4 Architecture and implementation

4.1 Overall architecture

Overall architecture of ePRO is shown in Figure 1. ePRO consists of three physical modules: target embedded system, digital multimeter, and host system.

We use TynuxBox [6] as our target embedded system. Embedded processor of TynuxBox is PXA255 which is based on XScale architecture [2]. The benefit of using XScale-based system is that we can use the performance monitoring unit (PMU) which is provided in the architecture level. We can monitor various system behaviors such as instruction/data cache efficiency, instruction fetch latency and so on with little overhead during the execution of embedded applications. Source-available embedded linux operating system is ported to TynuxBox. Slight modifications to kernel are needed to support performance profiling module.

To perform hardware-instrumentation energy profiling of embedded applications, we use a digital multimeter. The multimeter should be able to sample DC current at high frequency to increase the accracy of the tool. The multimeter should also be controlled by both embedded target system and host system to collect energy profiling via external trigger input and output. Currently, we use a Agilent 34401A digital multimeter.

We use a Pentium-4 desktop computer system as a host system. Linux 2.4.20 is installed on the host system. Host system and digital multimeter communicate with each other by using GPIB/Ethernet interface.

Logically, ePRO also includes three modules: energy profiler, performance profiler, and user interface. We will describe three logical modules in the following sections.

4.2 Energy Profiler

Energy profiler consists of system monitor, multimeter, and energy analyzer. We used the similar approach propsed by Flinn et al. [1] to design energy profiler. Energy profiling is performed in two-stages: data collection and data analysis. During data collection stage, system monitor periodically triggers a multimemter to collect DC current samples. Profiled DC current values are sent to host system via GPIB/Ethernet interface and saved on the file system of host system. System monitor also collects system information data such as program counter (PC) and process identifier (PID) with the same period. Unlike PowerScope, the daemon of the system-monitoring module was removed and the function call path for performance profiling was simplified to reduce the overhead incurred by energy profiler.

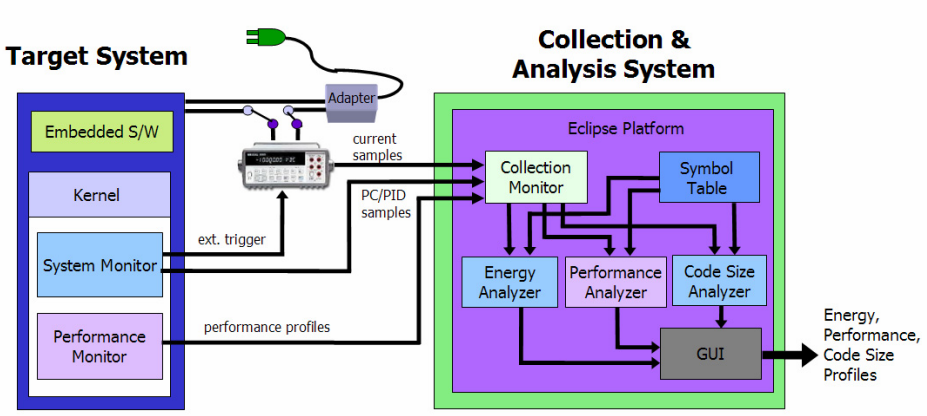

Figure 1. Overall architecture of ePRO

In data analysis stage, energy analyzer running on a host system analyzes both profiled DC current samples and system information data to generate energy profile. Using the symbol tables of the executables generated by cross-compiler, energy profiles can be mapped to specific procedures of embedded applications [1].

4.3 Performance profiler

Performance profiler consists of a code modifier and a small set of modifications to embedded Linux kernel. The overall flow chart of the performance profiler is shown in Figure 2. Code modifier takes the source code of embedded applications written in high-level language, such as C language as an input and produces modified source code of embedded applications as an output. Performance analyzing codes (PAC) are inserted into original source code in procedure level. Therefore, performance profiling can be performed in procedure level like profiling of energy consumption.

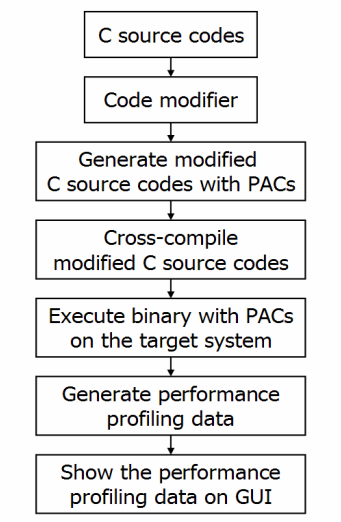

Figure 2. Overall flow chart of performance profiler

We also implemented a couple of system calls to embedded linux kernel. XScale has three performance monitoring counter registers and one performance monitoring control register. These registers can be accessed in privileged mode only, that is kernel-mode in linux operating systems. Therefore, we implemented several system calls with which we can access registers described above.

Once modified code with PACs is generated, we cross-compile the modified code and execute the binary program on the target system. Then, profiled performance data is saved on the file system of the target system. Using performance profiler, we can profile seven different kinds of system activity such as instruction cache efficiency, data cache efficiency, instruction fetch latency, data/bus request bus full, stall/writeback statistics, instruction TLB efficiency, and data TLB efficiency with little overhead.

4.4 User interface

Profiled energy consumption and performance data are shown by user interface. We designed user interface to meet two requirements. First, user interface should present profiled energy consumption and performance data in a user-friendly format. User interface shows profiled data in a graphical way so that programmers can easily identify the hot spots in the embedded applications. Another requirement is that user interface should be easily integrated with conventional embedded application development tools. We are currently integrating the user interface of ePRO with Eclipse [6] which is an open platform for tool integration built by an open community of tool providers. By integrating the user interface with conventional embedded application development tools, we can help embedded application programmers develop the embedded applications with high performance and low energy consumption more fast and easily.

5 Case study

For our case study, we have selected a frequency modified Fourier transform (FMFT) [7] program. Profiling

results generated by ePRO are shown in Figure 3. It can be easily seen that we need to optimized the `phifun` procedure, which computes phi-function in FMFT application, since it accounts for more than 70% of energy consumption of a whole program.

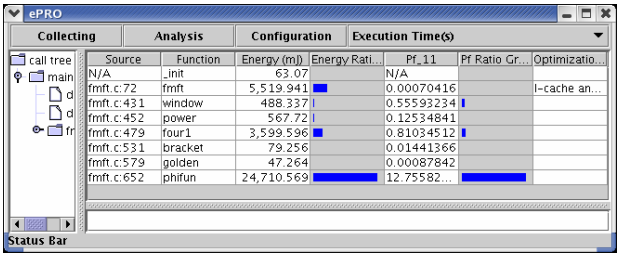

Figure 3. Profiling results for FMFT application

However, we still do not know why `phifun` procedure consumes such a large amount of energy. By profiling the performance of FMFT application using ePRO, we have figured out that the execution time of `phifun` procedure is more than 80% of the total execution time of FMFT application. We can infer that `phifun` procedure consumes much energy because it is called very frequently during the execution of FMFT application. This example clearly shows why performance profiling is important as much as energy consumption profiling.

Analyzing the original source code of `phifun` procedure of FMFT application, we recognized that it was implemented inefficiently for two reasons. First, `phifun` procedure uses unnecessary `malloc` and `free` functions in C language which are usually executed very costly. We simply eliminated them by using arrays in C language. Second, `phifun` used `sin` and `cos` functions which are implemented in the math library of C language. `sin` and `cos` functions are also very costly functions. We made lookup tables of sine and cosine functions and refered to lookup tables to reduce the overhead occured by calling `sin` and `cos` functions.

	Original	Optimized	Improvement (%)
E_{phifun}	24.71J	23.19J	6.0
C_{phifun}	5033124337	4791157114	4.5
E_{total}	34.58J	32.72J	5.4
C_{total}	5667714736	5422150091	4.4

Table 1. Experimental results for FMFT application

Experimental results are shown in Table 1. E_{phifun} and E_{total} mean the energy consumption in CPU due to `phifun` procedure and the entire program, respectively. C_{phifun} and C_{total} are CPU cycles taken by `phifun` procedure and the entire program, respectively. Before we optimized FMFT application, `phifun` procedure consumed 24.7J and executed in 5,033,124,337 cycles. After we optimized `phifun` procedure, it consumed 23.2J and executed in 4,791,157,114 cycles. Our naive optimization improved energy consumption and performance of `phifun` procedure by 6.0% and 4.5%, respectiviely.

6 Conclusions

Achieving high-performance and low-energy embedded applications is one of the most important issues in the design of embedded applications. To help programmers to develop efficient embedded applications in terms of energy and performance more easily, the demand for a tool which can map the profiled information of energy consumption and performance to program structure is increasing. In this paper, we describe the architecture and implementation of ePRO, a tool for energy and performance profiler for embedded applications. In our case study, we could improve energy consumption and performance of a DSP application by 5.4% and 4.4%, respectively using ePRO.

As our future work, we plan to improve ePRO by increasing the accuracy of the tool while reducing the overhead incurred by the tool. We also have a great interest in optimizing more complex and practical embedded applications such as MPEG4 encoder and decoder program using ePRO.

References

[1] J. Flinn et al., "PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications", In *Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, 1999.

[2] http://www.intel.com/design/intelxscale/

[3] D. Brooks et al., "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations", In *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2000.

[4] W. Ye et al., "The Design and Use of SimplePower: A Cycle Accurate Energy Estimation Tool", In *Proceedings of 37th Design Automation Conference (DAC)*, 2000.

[5] D. Shin et al., "Energy-Monitoring Tool for Low-Power Embedded Programs", *IEEE Design and Test of Computers*, 19(4), 2002.

[6] http://www.eclipse.org/

[7] http://www.boulder.swri.edu/~davidn/fmft/fmft.html