An Energy-Aware Whole-System Dynamic Emulator – SkyEye

Cheng Yu[1], Kang Suo[1], Ren Jie[1] ,Zhu Hui[2], and Shi Yuan Chun[1]
[1] Department of Computer Science and Technology
Tsinghua University, Beijing 100084, P.R.China
[2] Department of Computer Science and Technology
Beijing University of Aeronautics&Astronautics,
Beijing 100084, P.R.China
yuchen@tsinghua.edu.cn

Abstract. This paper presents the design of a high performance energy-aware whole-system emulator -- SkyEye. Several optimization and novel energy estimation methods used in SkyEye are proposed. By using novel searching strategy for Translated Block (TB), SkyEye save the time to find proper translated block. SkyEye uses Basic Equal Length Unit (B-ELU) method, and dynamic binary translation to reduce the simulation and energy evaluation time. The performance model of B-ELU is built to get the best length of translated block. In addition, the simulator automatically detects the voltage/frequency variation, and adjusts the energy estimation model accordingly. Using these methods, SkyEye which simulates ARM CPU based hardware system achieves marvelous performance and energy-aware statistic capability in experiments.

Keywords: Emulator, Dynamic Binary Translation, Embedded System, Energy Estimation

1 Introduction

Energy efficient embedded system design is becoming increasingly important with the proliferation of portable, battery-operated applications. The energy constraints on embedded systems are becoming increasingly tight as complexity and performance requirements continue to be pushed by user demand. Energy-aware simulator is a useful tool to develop energy efficient programs. Energy models for the target hardware platform are embedded in the simulator. When a program is developed in a high-level language and compiled, it is able to run on the simulator without any real hardware. And the energy consumed by this program can be estimated with high accuracy. But simulation speeds of these tools are very slow. Nowadays, Dynamic binary translation (DBT) and optimization technologies have achieved a high profile, since there are many famous projects such as the SimOS[1], Embra[2], DELI[3], IBM DAISY[6] open-source project, QEMU[4] open-source project, UQDBT[5], Transmeta Crusoe, and HP Dynamo. It's very appeal to combine DBT technology to speedup the energy evaluation time in simulator. Furthermore, the simulator will generate a detailed profile to describe energy consumption of each function in this

program. All these feedback information will return to developers and facilitate the improvement of software design.

This paper introduces a high performance energy-aware whole-system hardware emulator –SkyEye[7]. It can run an unmodified guest operating system (such as Linux, uClinux, eCos, L4) and all related applications. SkyEye itself runs on several host operating systems such as Linux, Windows. Currently, SkyEye is capable of simulating most ARM-based hardware platforms and various peripheral devices, such as NIC, LCD, Flash and UART. Compared with other similar emulators, SkyEye has the following desirable features:

1) Speed optimization;
2) Flexible combination of hardware simulation;
3) Automatic detection of task switches and function calls in OS;
4) Hardware statistic function;
5) Support for voltage scaling simulation..

2 Related Works

SimOS [1] is a MIPS based whole-system emulator. The key part of SimOS is Embra[2] which can run large, real-world programs and commercial operating systems. Embra can handle self-modifying code and can self-host. But the development of SimOS was stopped. The QEMU machine emulator and dynamic binary translator [4] is an emulation project that was started by Fabrice Bellard. It provides experimental implementations for x86, ARM, SPARC and PowerPC source machines running Linux.

Vivek Tiwari[9] develops an instruction level power model for Intel 486DX2 and Fujitsu SPARClite. The basic idea is the sum of the power costs of each instruction that is executed in a program can be estimated for the power cost of the program. Jeffry T. Russell[10] believes that there is actually no need to consider the individual assembly instructions to accurately predict energy and energy consumption. The energy consumption can be predicted by using the processor average power consumption multiplied by the software execution time. This easier method accurately predicts energy consumption to within 8% with 99% confidence based on physical measurements. Paper [11] presents an instruction class profiling technique, which has an estimation error of less than 3% with trivial runtime overhead. All these works are very successful, but most of these traditional works focus on the processor unit, not the overall system. The effect of a model on the overall system energy consumption is more important than its effect on the particular component it concerns.

3 System Architecture and Implementation

This section presents the SkyEye architecture and implementation of whole-system simulation. The main goal of SkyEye is to run/debug/analysis guest operating system on it. Figure 1 illustrates the structure of SkyEye. SkyEye is made of several components:

- CPU emulator (ARM, Blackfin, Coldfire, MIPS)
- Device emulator (LCD, Net IC, Flash, UART, Touch Screen, …)
- Dynamic Binary Translator/Tracer/Fixer (dynamic translating target binary code and optimizing translated host binary code)
- Debugger (debugging the OS on SkyEye in source level with GDB)
- Power Analyzer (analyzing the Power/Energy of Whole System consumption)
- Machine Configuration (according to the Machine Configuration file to assemble the simulated board-level hardware)

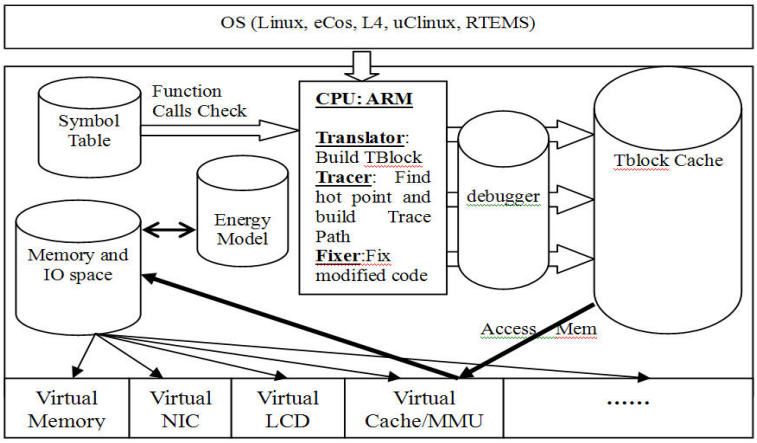

Fig. 1. SkyEye Structure

When SkyEye needs to estimate the Power/Energy consumption in the whole hardware system, then LDR/STR or other memory access instructions and the behavior of devices are estimated by "SkyEye Device and Memory Energy Model", while other normal instructions are estimated by "SkyEye Instruction Energy Model". SkyEye Symbol Tables Component records the function names and entry addresses. If PC register is equal to the entry address of function, Function Call Check Component is activated and detailed energy consumption information for that function is recorded.

4 Energy Evaluation and Optimization Technologies

4.1 Energy Model and Evaluation

The dominant source of power consumption in digital CMOS circuits is the dynamic power dissipation, which is computed by formula

$$P = C_a \times N_{sw} \times V_{dd}^2 \times f \quad (1)$$

where C_a is the output capacitance, N_{sw} is the number of switches per clock, $\mathrm{V_{dd}}$ is supply voltage, and f is the processor clock frequency. Because SkyEye is a instruction-level simulator, the leakage of current is ignored. Processor clock frequency is almost linearly related to $\mathrm{V_{dd}}$

$$\mathrm{f} = \mathrm{k} \times \frac{(\mathrm{V_{dd}} - \mathrm{V_{th}})^2}{\mathrm{V_{dd}}} \tag{2}$$

where k is a constant, and $\mathrm{V_{th}}$ is the threshold voltage. In practice, the voltage V_{dd} will decrease by slowing the clock, the energy consumption, which is proportional to the square of the voltage, would usually be reduced.

The energy, E, consumed by a processor running a program is

$$E = \int_0^T P(t)dt = \int_0^{\mathrm{T}} \mathrm{V(t)I(t)dt} \tag{3}$$

where T is the software execution time, $P(t)$ is the instantaneous power, $\mathrm{V(t)}$ and $\mathrm{I(t)}$ are instantaneous voltage and current. Average power, P_{avg}, is defined as

$$P_{avg} = \frac{1}{T}\int_0^T P(t)dt \tag{4}$$

and thus we can write

$$E = T \times P_{avg} \tag{5}$$

Accordingly, to measure energy consumption, a measurement for average power and total execution time is required. Total time, T, is related to N, the total number of execution cycles, and τ, the clock period, by $T = N\tau$.

The instruction set of ARM architecture is denoted as $ISA = \{A_1, \cdots, A_n\}$. A program can be deemed as a sequence of instructions during runtime, and the energy consumption of instruction A_j is calculated by

$$E_j = I_j \times V \times N_j \times \tau \times M_j \tag{6}$$

where E_j is the total energy consumption of all A_js in the program, I_j and N_j is the average current and execution cycles of A_j, V is the supply voltage, M_j denotes how many instruction A_j is executed in the program. In this way, we get a discrete format of formula (3)

$$E = \sum_{\mathrm{A_j} \in \mathrm{ISA}} \mathrm{E_j} \tag{7}$$

In order to simulate Dynamic Power Management (DPM) environment, a program is divided into several segments. Different segment may run at different processor voltage or frequency, but the voltage and frequency is constant inside one segment. In this paper, it is assumed that voltage scaling has a fixed discrete scope, defined as $\mathrm{S} = \{\mathrm{V}_1, \cdots \mathrm{V}_\mathrm{m}\}$.

Definition 1: Instruction Average Current Matrix (IACM) is a matrix to record the average current of each instruction at each supply voltage.

$$\mathrm{IACM} = \begin{pmatrix} \mathrm{I}_{11} & \mathrm{I}_{12} & \cdots & \mathrm{I}_{1\mathrm{n}} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \mathrm{I}_{\mathrm{m}1} & \mathrm{I}_{\mathrm{m}2} & \cdots & \mathrm{I}_{\mathrm{mn}} \end{pmatrix} \tag{8}$$

where I_{ij} denotes the average current of instruction A_j at voltage V_i.

Definition 2: Instruction Execution Time Vector (IETV) is a vector to record the total execution time of each instruction A_j in a program segment.

$$\mathrm{IETV} = (\mathrm{N}_1 \times \mathrm{M}_1 \times \tau, \mathrm{N}_2 \times \mathrm{M}_2 \times \tau, \cdots, \mathrm{N}_\mathrm{n} \times \mathrm{M}_\mathrm{n} \times \tau)^T \tag{9}$$

where $N_j (1 \le \mathrm{j} \le \mathrm{n})$ is the execution cycles of A_j, τ is the clock period, M_j denotes how many instruction A_j is executed in the program segment.

Definition 3: Voltage Selection Vector (VSV). Since different segment may adopt different supply voltage, VSV describe which voltage is selected by current segment.

$$VSV = \{0, \cdots, V_k, \cdots, 0\} \tag{10}$$

VSV has m elements, but only one can be nonzero. If the k th element is nonzero, the k th element must be equal to V_k, which means the current segment select supply voltage V_k.

According to the theory explained before, the energy consumption of one program segment can be estimated by the segment's IETV and VSV.

$$E_{seg} = VSV_{seg} \times (IACM \times IETV_{seg}) \tag{11}$$

And the total energy of the whole program in a dynamical power management environment is estimated by

$$E = \sum_{seg} IACM \times IETV_{seg} \times VSV_{seg} \tag{12}$$

In formula (12), IETV and VSV is online recorded by simulator during runtime, IACM is provided by offline measuring the instruction current on a specific target hardware platform.

In this energy model, even the same instruction may have different energy consumption with different arguments. For example, LDR/STR instruction can access

I/O address space, which usually involves various peripheral devices to work and the energy consumption is varied greatly. In previous related research work, this problem is not resolved thoroughly because the simulation to peripheral devices is usually not implemented.

In SkyEye, several peripheral devices are simulated and multiple device states are defined for each of them. Different device state has different power consumption, and is assigned a different power consumption model. The transitions between states are assigned an fixed energy cost value, which is obtained by experiments on target hardware. If a program segment involves some operations on the peripheral devices, the total energy consumed is estimated by summing the energy of all the involved devices and transitions. This model is suitable for the energy estimation in DPM environment.

We run three applications on linux-2.6.x kernel to compare their energy cost. In the first two applications, we turn the DPM on. In the third one, we turn the DPM off . The Linux kernel is modified to support DPM. The CPU frequency will be decreased to save the energy when the Linux system is in its idle state. When the CPU is wake up, it returns into busy computation, and the CPU frequency will return to the highest value. Table 1 displays the instruction number, cycles and energy computed by SkyEye. And Figure 2 compares the energy estimated by SkyEye and the energy measured on the Sitsang board, which is based on Intel PXA25x CPU. The experiment results show that the estimation error of SkyEye is kept within 20%.

Table 1. Three Applications in DPM Environment.

	APP1	APP2	APP3
Instructions	25175079	25053189	25095178
Cycles	55148607	54821274	67557386
Energy (J)	0.097121	0.096557	0.105776

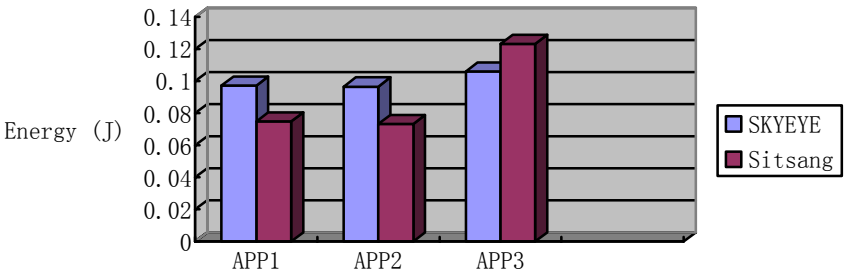

Fig. 2. Comparison on SkyEye and hardware

4.2 Dynamic Binary Translation

When SkyEye first encounters a piece of target CPU codes, SkyEye Translator Component dynamic translates the codes into translated block (TB). When SkyEye use basic translating method, the length of each target CPU codes is equal. Along with the OS&applications run on SkyEye, the SkyEye Tracer Component monitors the executive frequency of TB, finds the hot spot, and combines several TBs in hot spot executive path into a bigger Super TB adaptively. SkyEye Fixer Component monitors the self-modifying code, and invalidates corresponding TB to guarantee the consistency. If a given TB is invalidated too often because of write accesses, then a bitmap representing all the code inside the TB is built. Each time when CPU writing into that TB, SkyEye Fixer Component will checks the bitmap to see if the code really needs to be invalidated, so SkyEye Fixer Component avoids invalidating the code parts in TB when only data is modified in TB. In order to improve the performance of SkyEye, we designed several new methods in SkyEye Translator/Tracer/Fixer. The sections below will show these details.

4.3 Performance Model of Translated Unit Constructing

The general methods for translating and constructing unit are Basic Block method and Trace method. Because SkyEye emulates embedded RISC CPUs for which each instruction has the same length, SkyEye use another method — Basic Equal Length Unit method (B-ELU Construction) to build the translated unit. The B-ELU construction method dynamically translates one piece of target code with the same length each time, so it is much simpler than Basic Block or Trace method, and the construction time for the translated unit is less than the other two methods. The key factor of B-ELU method is to choose a suitable instruction length for translated unit.

There are four TB accessing mode in the process of SkyEye B-ELU translation. Mode I: Query the recent accessing record of TB, get the corresponding entry address of TB; Mode II: Query the List/Hash Table of translated block, get the corresponding entry address of TB; Mode III: Query the List/Hash Table of translated block, get the corresponding entry address of TB; Mode IV: Translate block at the first time, get the corresponding entry address of TB and update List/Hash Table and recent accessing record.

B-ELU method can split the Text (Code) Section Space of user-level process or OS into N (N>=1) equal length TBs. If the length of TB is very short, then SkyEye translator has to spend more time on switching the execution of TB and the translating of TB will decrease. But if the length of TB is very large, some problems will rise:

1. There may exist many accesses whose entry address aren't at the beginning or end of the TB, so SkyEye Translator Component will spend more time on to partial translation of TB.
2. There may have several non-instructions in large TB, so SkyEye Translator Component has to do several unnecessary translations.
3. Large TB could include Text Section, Data Section, BSS Section, so SkyEye Fixer Component for self-modified code might make wrong judgment, and do some unnecessary TB re-translation.

In order to find the best Search strategy for TB, we design and compare three different search strategies for TB[12]. The primary concern is how to reduce searching time of access address for TB. The first strategy is to record the translated TB in a list, and search the list every time. The second strategy is to use hash table to replace list and use recent access address record to store the last access address of TB. The third strategy is to record all possible access address in a translated TB and the last access address of TB. The speed of each strategy is: Strategy 1 < Strategy 2 < Strategy 3.

In order to find the suitable length of translated block (TB), the ELU performance model was introduced [12]. The main executive cycle of SkyEye includes: Searching TB----Translating TB----Executing TB. We did several measurements by running uClinux and MiniGUI applications on SkyEye, and get those results from experiments。

It shows that the search strategy 3 is SkyEye's best B-ELU search strategy. According the experiments, the best region of the length of TB is {256, 4096}.

From the SkyEye B-ELU performance formula, we can analyze the factors that affect the translation performance. When we use B-ELU with best strategy and Best Length of TB, the result is very good. But we also have detected that the time of switch time of TB executing turns into the performance bottleneck of SkyEye.

5 Experiments and Results

The Experiment environment includes a PC with 2GHZ P4 CPU with 256MB memory. We run uClinux-2.6.x+applications on SkyEye with different strategies and different length of TB. We also have modified QEMU-0.8.0 to run ARM Linux-2.6.x+applications and test the performance of QEMU. Figure 3 compares the performance of SkyEye using different methods and QEMU. SkyEye and QEMU were compiled by GCC-3.3 with CFLAGS='-pg …', then the performance statistic data can be gathered by gprof.

From the data collected by gprof, SkyEye using B-ELU with 4 bytes TU is about 1 times faster SkyEye using interpreting methods. If SkyEye chooses 4096 byes TU and searching strategy 3, it is about 6 times faster than SkyEye using interpreting methods. The speed if SkyEye is a little slower than that of QEMU. We also noticed that SkyEye used almost all memory (196MB) in PC for TB cache, but QEMU spend about 64 MB memories. Because SkyEye cache almost all translated blocks in memory, but QEMU just cache the recent translated blocks.

The experiment results show that QEMU uses optimized methods including basic block as TB, direct block chaining, condition code optimization and register allocation. The performance of QEMU was very good.

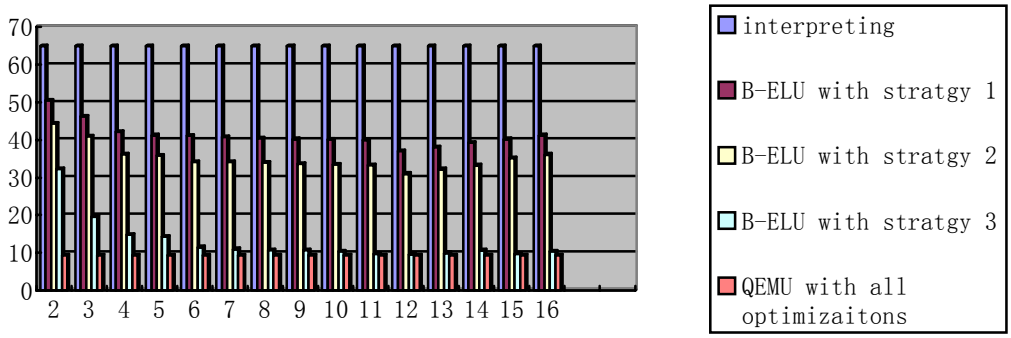

Fig. 3. the performance of translating&optimizing methods in SkyEye and QEMU. The value of X axis: the length of TB= 2^(x) using B-ELU in SkyEye. The value of Y axis: executing time (second)

6 Conclusions

In this paper, we present an implementation of a emulator—SkyEye. SkyEye uses novel energy model for ARM CPU based hardware platform. The new energy model supports DPM and considers the problem of peripheral devices. SkyEye uses novel searching strategy for TB to reduce the search time. The performance formulas of dynamic binary translation were built and the most suitable TB length was calculated using measured parameters. With the above strategy and methods, SkyEye got a good performance in experiment The future work will focus on design adaptive block linking (ABL) method to delimit the unnecessary switch time of TB executing, designing the distributed SkyEye Tracer to improve the performance of SkyEye further, designing loop structure recognizer to optimize the loop structure in target OS&Application. The debugger support on SkyEye will be added to help developers to debug system software more easily.

Acknowledgments. This paper was Supported by the University. Doctorial Research Foundation, No.20050003048.

References

1. Rosenblum, M., Herrod, S., Witchel, E., AND Gupta, A. 1995. The SimOS approach. IEEE Parallel and Distributed Technology, 4(3), 34-43.
2. Witchel, E., AND Rosenblum, M. 1996. Embra: Fast and flexible machine simulation. In Proceedings of the 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 68-79.

3. Desoli, G., Mateev, N., Duesterwald, E., Faraboschi, P., AND Fisher, J. A. 2002. DELI: A new run-time control point. In Proceedings of the 35th International Symposium on Micro architecture (MICRO '02), 257–268.
4. David Ung and Cristina Cifuentes. Optimizing Hot Paths in a Dynamic Binary Translation. Workshop on Binary Translation, 2000. 38- 139
5. Fabrice Bellard. The QEMU CPU Emulator, 2004. http://fabrice.bellard.free.fr/qemu/.
6. Kemal Ebcioglu and Erik R. Altman. DAISY: Dynamic Compilation for 100% Architectural Compatibility. In ISCA, pages 26–37, 1997. 41
7. Chen Yu, etc., SkyEye Emulator, http://www.skyeye.org
8. Kang Shuo, Wang Hua yong, Chen Yu, “An Energy-awared Simulator for Energy Co-estimation in the Embedded System”, The 2004 International Conference on Embedded Software and System, LNCS.,2004.10
9. Vivek Tiwari, Sharad Malik and Andrew Wolfe, "Power Analysis of Embedded Software: A First Step Towards Software Power Minimization", IEEE Transaction on VLSI Systems, Vol. 2, Issue 4, Dec 1994.
10. Jeffry T. Russell and Margarida F. Jacme, "Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors", In Proc. of International Conf. on Computer Design: VLSI in Computers and Processors(ICCD'98), pp. 328-333, 1998.

11. Amit Sinha, Nathan Ickes and Anantha P. Chandrakasan, "Instruction Level and Operation System Profiling for Energy Exposed Software", IEEE Transaction on VLSI Systems, Vol. 11, Issue 6, Dec 2003.
12. Chen Yu, Ren Jie, Zhu Hui, and Shi Yuan Chun, “Dynamic Binary Translation and Optimization in a Whole-System Emulator -- SkyEye”, Technique Report, Tsinghua University, 2006