Coordinating System Software for Power Savings

Lingxiang Xiang, Jiangwei Huang and Tianzhou Chen
College of computer science, ZheJiang University
ZheDa road 38#, Hangzhou, China, 310027
{lxxiang,hjw,tzchen}@zju.edu.cn

Abstract

Power consumption is becoming a primary concern as a result of tremendous increasing in computer power usage. Innumerable methods and techniques have been exploited to address this problem but few concentrate on collaborative approaches. This paper presents coordination mechanisms that integrate operating systems with compilers under power reduction techniques such as DPM and DVS. By remaining the information generated at compile stage about an application's structure and performance characteristics as much as possible and committing it to system software at run stage, the system software, especially the operation system and compiler, are collaborating toward file-grain power optimizations. The proposed coordination mechanisms also make it possible to integrate the power optimization approaches in our prior work into a whole system. Thus, the optimizations working at distinct levels can be overlaid at run time, and the power reduction effect can be enhanced.

1. Introduction

Over the past several years, power savings and optimizations has become an area of very active research as a result of tremendous increasing trend in computer power usage. Innumerable methods and techniques have been exploited for battery-powered portable systems, desktop computers and even servers. These technologies are either hardware-level or software-level directed. The former mainly includes circuit and logic, low-power interconnections, low-power memories and processor architecture adaptive technologies [1], while the latter are typically combined with hardware-level techniques.

Software directed approaches have more direct knowledge about the workload and accordingly have more control on large portions of the computer system due to their higher abstraction levels. Researches of software directed power optimizations focus on system software especially the operating system and compiler, which usually works together with two well studied power reduction techniques, dynamic power management (DPM) and dynamic voltage scaling (DVS).

Though a lot of software directed methods have been exploited, few research focus on the collaborative methods. Furthermore, the prior combination methods of system software are very rough and coarse-grain. For instance, in a recent study [18], the compiler is just used to partition code regions or insert some DVS instructions. The characteristic of being conversant with the structure of applications, which is the most important feature of the compiler, is not well utilized by the systems.

The motivation of this paper is to explore more opportunities for power optimization by coordinate components working at various levels to take full benefits provided by compiler and OS. To achieve this goal, this paper presents coordinate mechanisms that integrate operating systems with compilers under some power reduction techniques such as DPM and DVS. The

information concerning an application's structure generated at compile stage is collected as much as possible and then committed to the OS or dynamic compiler at run stage for further analysis.

By deploying proposed coordinate mechanisms, the system software, especially the operation system and compiler, are collaborating towards fine-grain power optimizations. The proposed coordinate mechanism makes it possible to integrate the power optimization approaches in our prior work into one system in which different methods cooperate efficiently to reduce computer power.

The rest of the paper is organized as follows: Section 2 summarizes related work. Section 3 gives an overview of our software directed power reduction system architecture and the coordinate mechanisms. Section 4 further develops in sequence more specific descriptions of power-aware system architecture which incorporates the scheduling related, device related and runtime coordinate mechanisms. This is followed by the experimental results in Section 5. Finally, Section 6 concludes the paper by summarizing our major contributions.

2. Related Work

From the perspective of the OS, power aware schedule policies are commonly used to satisfy the deadline or performance requirement of an application [2, 3] and resource hibernation policies have been adopted for hardware devices such as disk driver and cache [4].

Compiler-level power optimizations originally appeared in traditional compiler, among which reordering instruction to reduce switching [5, 7] and reduction of memory operands through register pipelining [6] are two typical techniques. After that due to the proposal of DVS, a new analysis model, dynamic compiler techniques, emerged to adapt its very feature [8]. Both static compiler techniques and dynamic compiler driven DVS techniques have been well developed [8, 9].

There are also several practices concerning the collaborative methods. Azevedo et al. [18] proposed a novel intra-task DVS technique based on compiler control which makes use of program checkpoints that carry user-defined time constraints. Checkpoints are generated at compile time to indicate where in the code the processor speed and voltage should be re-calculated. OS handles multiple intra-task performance deadlines and modulates power consumption according to a run-time power budget given by the user. In a recent study, AbouGhazaleh et al. [19] uses the compiler to annotate an application's temporal behavior information. This path-dependent information is then passed on to the OS as guidelines on how to periodically change the frequency of the processor during runtime.

It is obvious that the combination of compiler and OS is very rough in prior work where compilers are just used to partition code regions or insert some DVS instructions. The most important feature of compilers, being conversant with the structure of applications, is not well utilized by OS. So some potential opportunities for further power optimization are missed.

3. Architecture

3.1. An Overview

System programs play a critical role in the entire life cycle of an application, especially the operating system and compiler. To run a program, the compiler at first transfers the source code to machine dependent binary. Then, the application is loaded

and waits to be selected by the OS scheduler. Once chosen to be an active running application, the application runs upon the operating system and interacts with it. At the same time, dynamic compilers collected the application's runtime information based on which further modifications were applied to the application to improve its performance.

We have proposed a few power saving methodologies for almost all the phases of an application's life cycle [11, 12, 14, 15, 16]. In this paper, we integrate them into one system whose architecture is illustrated in Figure 1.

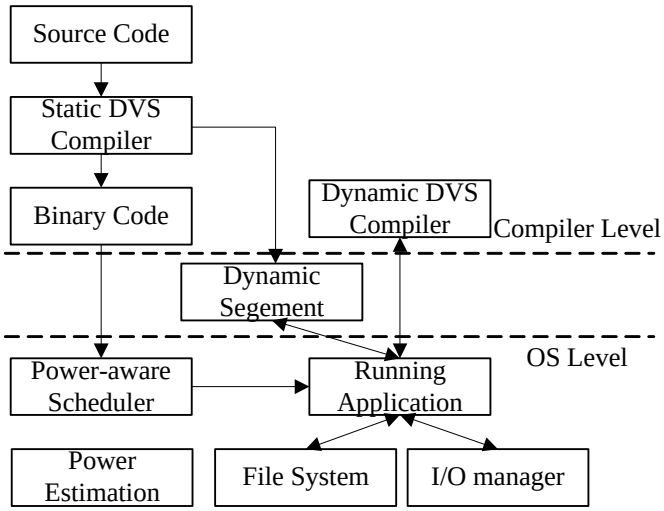

Figure 1. The architecture of software-directed power optimization system

According to the locations of the power optimizations in the system, they can be classified into three categories: compiler level, operating system level and collaborative level.

In compiler level, both traditional static compilation and recent dynamic compilation techniques that modify the binary code of a running application are introduced. The static compiler gives the information that can present an application's internal structure. At the later phases, other components also have a good knowledge of an application's structure via importing this generated information. The dynamic compiler works with applications simultaneously, so compared to the static compiler, it has further knowledge about real running environment and performance of the application, which makes it very useful to help conduct dynamic adjustment.

At OS level, the power estimation module [10] gives the basic power information of the entire application or specific code regions. What characterizes it from normal processor scheduler is that this optimized module puts the energy efficiency as the first consideration on determining how to execute a given application. Meanwhile, the power-aware file system and I/O manager can give further power optimizations.

We have also exploited a collaborative OS and compiler power management approach for embedded system by partitioning the program into different regions.

We have elaborated an overall schema of this collaborative power saving architecture, now the primary issue lies on how to let these components which locate at different levels

work coordinately and take full benefits from compiler and OS. We give details about the coordinate architecture mechanisms in the next section.

3.2. Coordinate Mechanisms

The compiler and OS interact with an application at different stages, thus they have different knowledge about the application. Taking the whole application as input, the classic compilers are highly informed with the structure of applications. It can partition the application into different regions and estimate the static properties of these regions. It also gives well considerations to the I/O and storage demands. But it lacks the capability of predicting future execution behaviors due to lack of running environment information. In contrary, OS can estimate and predict the power/performance behavior, device visiting profile and file system operations since it controls the entire running environment.

To fully utilize the features of compiler and OS, our proposed coordinate mechanisms consist of follow aspects.

- The OS estimates the power consumption of regions partitioned by compiler according to CPU workload [10]. It then evaluates the power consumption of applications and use power saving schedule algorithms for task scheduling. At the same time, the application's real-time requirement is strictly guaranteed. (Refer section 4.1 for details)
- Based on the I/O demands gathered at compile stage, the OS schedules the device operations under specific policies such as flow model assisted DPM.(Section 4.2)
- External storage devices have high power consumption. Upon the power aware storage architecture included in our system [11], the OS optimizes the file system operations by employing the compiler's predication of the file visits.(Section 4.2)
- With the run-time performance provided by OS, dynamic compiler applies further power optimizations such as code rearrangements to running applications.(Section 4.3)
- At the run-time stage, the compiler-based regions structure and performance information are used collaboratively. Thus, the regions that have similar power/performance behavior can be merged together, while the large region can be split into small regions for fine-groan optimizations.(Section 4.3)

Overall, the primary goal of the coordinate mechanisms is integrating operating systems with compilers under two of the most efficient power reduction techniques, DPM and DVS. The information concerning an application's structure generated at compile stage is collected as much as possible and then passed to the OS or dynamic compiler at running stage for power optimizations.

4. Implementation

4.1. Scheduling Related Coordination

A real time DVS schedule algorithm developed in our prior work [12] is introduced into the system for random task schedule model. The appropriate frequency is obtained through the event-counters. Firstly, we get the relationship between the rates at which countable events happen at a certain time and the behavior of a system concerning performance and energy. An appropriate frequency that minimizes the energy consumption and arrives a given performance

requirement is determined by a periodic evaluation of the event rates in the latest history of the system. Then, the OS scheduler finds the frequency domain that matches all the event rates of the system. At last, it calls DVS instructions to adjust the CPU frequency.

4.2. Device Related Coordination

Dynamic Power Management via Compiler Assisted I/O Predication: I/O operations (*i.e.* file accesses) make up the major part of interactions between applications and external devices. By increasing the burstiness of device accesses and idle periods [13], this arrangement of I/O operations can lead to significant power reduction. Thus, the device can be turned into low power state during idle periods.

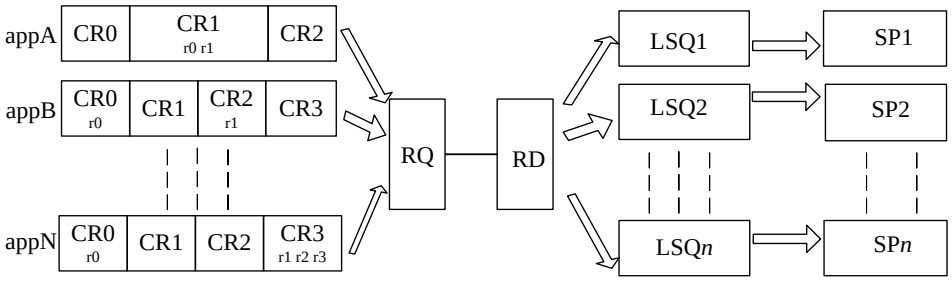

Figure 2 Multi-server and multi-requester system

Our system for energy efficient I/O is illustrated in Figure 2. It is a typical complex power-managed system that contains multiple running applications (appA, appB and so on) and multiple *service provider*s (SP, in the system, they are specialized as I/O service providers).

In each application, the code regions are partitioned statically by the compiler according to different I/O behaviors at compile stage. At the same time, the compiler also analyzes the power/performance characteristics of each I/O request and associates its power/performance property with the code region. For instance, in Figure 2, there are three *code region*s (CR) in appA where CR1 have two I/O requirements (r0 and r1). In appB, r0 and r1 exist in CR0 and CR2 correspondingly. All of these I/O requests are predicted by OS according to the current PC in the active process. The requests-to-taken are then sent to a *request queue* (RQ) which communicates with *request dispatcher* (RD) implemented as a kernel model.

The RD decides which SP should service which I/O request according to the power/performance parameters associated with the request. Once decided, the request is pushed into the *local service queue* (LSQ) that buffers the I/O requests for the SP.

Power-aware File System: The static complex information about file I/O can be applied to a lower level such as file system. While the hybrid large storage architecture [11], which uses the low power cost memory such as Compact Flash as the cache of a large capacity Microdrive in order to take advantage of lower cost large capacity and energy consumption simultaneously, is servicing as storage device, the file system utilizes the I/O information provided by compiler to place most frequently accessed files onto Compact Flash of low energy consumption and to place the others onto Microdrive. As a result of efficient schedule of file operations, most file reads take place on the compact flash, so that the Microdrive is in the standby mode in most of

the time. It spins up only when a file resident in the Microdrive is accessed. Meanwhile, all the writing operations are sent to the compact flash to keep the data in consistent.

4.3. Runtime Related Coordination

Dynamic Compiler Driven DVS: Our dynamic compiler driven DVS framework consists of the First-time Compile module (FC), the Monitor and Judger module (MJ), the Run-time Optimizer and Compiler module (ROC). As we have presented before, the basic unit of the DVS control is a region or a block of instructions [14]. In our previous work of static DVS control, we select loop and function unit as the candidates and insert the DVS instruction before and after these two types of regions. Now we evaluate these regions in the MJ module instead.

At the start, the FC module compiles the original codes for the first time and delivers it to the operating system and hardware. At the same time, this module selects the loop and function blocks or regions as DVS candidates, passes the basic information of the candidates to the MJ module. Meanwhile the OS and hardware run the code at the first time and collect the information of OS and hardware for the MJ module. Given all the necessary information, the MJ module works to determine the detailed method of the optimization which the ROC module conducts to optimize and recompile the codes.

Then the MJ module delivers these information and methods to ROC module, which performs the detail optimization including instruction rescheduling and DVS instruction inserting.

Dynamic Split and Combination of Code Regions: More opportunities for power optimization are available via partitioning the application into small code regions at compile time than treating the entire application as the target of optimization, to achieve a finer grain optimization. We import the methodology in [15, 16] that splits and combines the code regions dynamically to our system.

We use a modified compiler to scan the source code and find the desired regions. A program region is a function or a loop. Once a region has been found, the “Regions partition” model in the modified compiler will call the “Insert Region API Call” module to insert the Enter/Exit region API which communicates with the OS kernel when entering or exiting a region at run-time.

When the program is running, these regions are partitioned or merged dynamically according to the performance statistic which is gained by a kernel module. Then, different regions are assigned different CPU settings and executed on different CPU frequencies according to their characteristics.

5. Experimental Results

We have separately evaluated for the specific parts of this system from low level to high level on the Intel Xscale PAX255 platform. The power-aware file system, in contrast to the original develop board with only flash memory, caches more than 90 percents of file accesses in low power flash in most cases. It provides 32 times storage capacity with only 10-15% battery runtime loss. The power-aware scheduler achieves energy savings about 10% with little performance loss for some situations, while the compiler assisted

DPM policy with I/O predication can save more than 12% power compared to traditional DPM policies.

At the level above the OS, the CPU energy savings of dynamic compiler framework are in the range of 13%-15% with a permission of performance penalty of 0.5%-5.5%. And splitting or combining the code regions at run time leads to more power savings than using static method solely [15].

To fairly evaluate the power saving effect and to eliminate the interference from other parts, we have to evaluate each methodology solely. But it is notable that some power optimizations in the coordinate system can be overlaid at run time and the power saving effect will be enhanced. It is expectable that the gained benefits of our system would be much more significant in a real environment.

6. Conclusion

To explore more opportunities for power optimization, this paper integrates the power saving methodologies that work at almost all the phases of an application's life cycle together into one system. By deploying the coordinate mechanisms presented here, the system software, especially the operation system and the compiler, are collaborating towards fine-grained power optimizations.

The experimental results show that the collaborative approaches achieve significant power savings. We feel the proposed soft architecture and coordinate mechanisms have a great potential in addressing the power control and energy reduction problem.

7. Reference

[1] Sato, T. and Morishita, D. A field-customizable and runtime-adaptable microarchitecture. *Field-Programmable Technology (FPT)*, 2003.
[2] LORCH, J. AND SMITH, A. Improving dynamic voltage scaling algorithms with PACE. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. 50–61. 2001.
[3] Canturk Isci, Gilberto Contreras and Margaret Martonosi. Live, Runtime Phase Monitoring and Prediction on Real Systems. In *MICRO-39*. 2006.
[4] GURUMURTHI, S., SIVASUBRAMANIAM, A., KANDEMIR, M., AND FRANKE, H. DRPM: Dynamic speed control for power management in server class disks. *SIGARCH Comput. Architect.* News 31, 2, 169–181. 2003.
[5] G. Semeraro, D.H. Albonesi, S.G. Dropsho, G. Magklis, S. Dwarkadas, and M.L. Scott. Dynamic frequency and voltage control for a multiple clock domain microarchitecture. In *Micro-35*, 2002.
[6] Stefan Steinke, Ruediger Schwarz, Lars Wehmeyer, Peter Marwedel. Low power code generation for a RISC processor by Register Pipelining. *Technical Report 754, University of Dortmund, Dept. of Computer Science XII*, 2001.
[7] C. L. Su et al. Low power architecture design and compilation techniques for high-performance processors. In *IEEE COMPCON*. 1994.
[8] C-H Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. In *PLDI*, 2003.
[9] Wu, Q., Martonosi, M., Clark, D. W., Reddi, V. J., Connors, D., Wu, Y., Lee, J., and Brooks, D. A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance. In *MICRO-38*.2005.
[10] Huang Jiangwei, Chen Tianzhou, Qian Jie, and Liang Xiao. Power Estimation for an Application on the Xscale Platform Using PMU events. In *International Conference on Wireless and Mobile Communications and the International Multi-Conference on Computing in the Global Information Technology*, ICISP/ICDT. 2006.
[11] Chen Tianzhou, Yan Like and Xie Bin. An Implementation of Power-aware Storage Architecture. In *IWCMC*. 2006.
[12] Chen Tianzhou, Qian Jie, Shi Qingsong, Huang Jiangwei. A New Dynamic Frequency Scaling Algorithm for Power-Aware and Real-Time System. In *Eighth Real-Time Linux Workshop*, 255-260. 2006.

[13] Nathuji, R., Seshasayee, B., and Schwan, K. Combining compiler and operating system support for energy efficient I/O on embedded platforms. In SCOPES '05. 2005.
[14] Chen Tianzhou, Liang Xiao, Huang Jiangwei. Energy saving compiler framework on embedded systems. *IASTED International Conference on Power, Energy and Applications*, PEA 2006.
[15] Huang Jiangwei, Chen Tianzhou, Qian Jie, Liang Xiao. Partitioning the Program into different regions Using dynamic and Static Approach with kernel-Assisted in Power management for Embedded System. In *The 2006 IEEE International Conference on Information Reuse and Integration*, IRI-2006, 341-344. 2006.
[16] Huang Jiangwei, Chen Tianzhou, Qian Jie, Liang Xiao. Using Dynamic and Static Approach with Compiler-Assisted in Power Management for DVS-Enabled Embedded System. In *WSEAS TRANSACTIONS on COMPUTERS*, Volume 6, Issue 6, 2007.
[17] Azevedo, A., Issenin, I., Cornea, R., Gupta, R., Dutt, N., Veidenbaum, A., and Nicolau, A. Profile-Based Dynamic Voltage Scheduling Using Program Checkpoints. In *Proceedings of the Conference on Design, Automation and Test in Europe*. 2002.

Authors

Tianzhou Chen obtained his B.S. degree in computer software in 1994. He studied for M.S. degree in computer architecture, and received his PH.D degree in computer application from Zhejiang University in 1998. He is a professor of computer science at Zhejiang University. And he is the vice director of computer systems engineering research institute of Zhejiang University. He is a member of IEEE, and ACM. His current research interests include computer architecture, multi-core system, on-chip interconnection, embedded system, power-aware computing, hardware/software co-design and security. He has authored and/or coauthored 12 textbooks, 161 research papers in major journals and international conferences in computer architecture area. He holds 27 Chinese Patents, 49 Software Copyrights.