Third 2008 International Conference on Convergence and Hybrid Information Technology

Low-cost Application-aware DVFS for Multi-core Architecture

Joonho Kong[1], Jinhang Choi[1], Lynn Choi[2], and Sung Woo Chung[1]
Department of Computer & Communication Engineering[1]
School of Electrical Engineering[2]
Korea University
{luisfigo77, cepiross, lchoi, swchung}@korea.ac.kr

Abstract

As technology scales down, energy/power consumption in the microprocessor has become a serious problem. Especially, as the industry moves on to multi-core processor systems, energy/power management in multi-core systems has become more and more important. In this paper, we propose new DVFS technique in multi-core systems. Our proposed technique finds the optimal DVFS level using Energy-Delay Product (EDP) and Energy-Delay^2 Product (ED^{2}P), which considers energy-efficiency of computation. According to determined DVFS level, the voltage and frequency of processor cores are changed. In our evaluations, our proposed technique shows 5.6% and 54.3% EDP reduction compared to the energy-biased and performance-biased scheme, respectively. In case of ED^{2}P, our proposed technique reduces 36.4% and 14.7% of ED^{2}P compared to the energy-biased and performance-biased scheme, respectively. The proposed technique can be a good alternative in future multi-core system where the both energy/power consumption and performance are critical.

1. Introduction

As technology scales down, energy/power consumption in the microprocessor has become a serious problem [1]. Excessive energy consumption gives many serious effects to the microprocessor such as battery life or temperature. Furthermore, excessive energy/power consumption also affects processor's performance. For this reason, the microprocessor cannot operate well in given energy/power budget due to energy/power management schemes. Therefore, both reducing energy/power consumption and retaining the performance are crucial in the microprocessor design. There should be a trade-off between performance and energy/power consumption.

To overcome the energy/power problem, many techniques are proposed. Among these techniques, Dynamic Voltage and Frequency Scaling (DVFS) is simple and effective technique for energy reduction, which is already implemented in modern microprocessors [12]. It switches processor's voltage and frequency dynamically according to processor's power consumption or thermal condition. DVFS controller dynamically adjusts microprocessor's voltage and frequency to reduce energy/power consumption.

On the other hands, due to the limitation of Instruction-Level Parallelism (ILP) [8], microprocessor designers moved on to the multi-core microprocessor design. Although multi-core design cannot improve latency of the microprocessor, it improves the throughput of the microprocessor significantly. However, excessive energy consumption in multi-core microprocessors is also a serious problem. Thus, as moving on to the multi-core era in the microprocessor design, energy/power management in multi-core architecture has also become important. Especially, it becomes more important in mobile/embedded computing environment where the energy/power consumption is a critical factor. Furthermore, multi-core architectures such as ARM11 MPCore [11] become a main stream of mobile/embedded environment.

We propose an efficient energy management technique, which leverages the trade-off between energy/power consumption and performance. We use DVFS for energy/power reduction scheme, which can manage energy/power consumption adaptively. According to the Energy-Delay Product (EDP) of each application, the microprocessor determines voltage and frequency of each core dynamically. As adapting voltage and frequency according to EDP, it can achieve

978-0-7695-3407-7/08 $25.00 © 2008 IEEE
DOI 10.1109/ICCIT.2008.124

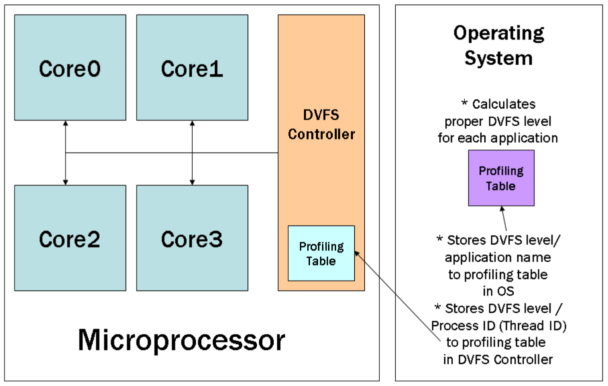
Core0
Core1
Core2
Core3
DVFS Controller
Profiling Table
Microprocessor
Operating System
* Calculates proper DVFS level for each application
Profiling Table
* Stores DVFS level/ application name to profiling table in OS
* Stores DVFS level / Process ID (Thread ID) to profiling table in DVFS Controller

Figure 1. System architecture framework (Calculating DVFS level is required only once per each application.)

both significant energy reduction and less performance loss.

The following sections are organized as follows. Section 2 describes our proposed technique. Section 3 shows the evaluation results. Lastly, we will conclude this paper in Section 4.

2. Adaptive DVFS for multi-core architecture

2.1 Conventional design

Conventional energy/power management scheme in multi-core processor is to adjust voltage and frequency level according to temperature or energy consumption [4]. It is the most widely used scheme in modern microprocessors. However, conventional energy management scheme inevitably has performance loss. Moreover, recent study showed that malicious thread can intentionally degrade the performance of normal threads using excessive energy/power density [5]. It implies conventional energy/power management scheme has many disadvantages in terms of performance. Thus, energy/power management while maintaining performance is crucial in the microprocessor design.

Table 1. Voltage and frequency for each DVFS level

	Level0	Level1	Level2	Level3
Voltage	1.0V	1.3V	1.6V	1.9V
Frequency	0.6 GHz	1.2 GHz	1.8 GHz	2.4 GHz

2.2 New DVFS technique for multi-core architecture

In multi-core processors, many processes or threads are executed in parallel. By exploiting parallelism, multi-core processors can maintain performance though they use reduced frequency and voltage compared to superscalar processors. Since each process (threads) has their own application features, we can exploit their features to minimize performance loss.

2.2.1 Overall system architecture framework

We assume the typical multi-core system which has four cores in a single chip. In our proposed technique, proper interactions between operating system and DVFS controller are required. Overall system framework is depicted in Figure 1.

Operating system plays a key role in our proposed technique. When context switching is incurred, OS calculates DVFS level for newly fetched application using program behavior analyzing tools and gives information (Process/Thread ID and Optimal DVFS level) to profiling table in DVFS controller and OS. We use two profiling tables. The data in profiling table in DVFS controller is removed when the microprocessor is turned off. In this case, we should calculate DVFS level of every application whenever the system is rebooted, which is significant overhead. Thus, we use one more profiling table in OS. When the application is once executed, OS calculates DVFS level and store to the profiling table in OS (non-volatile) as well as in DVFS controller (volatile). Consequently, calculation is required only once for each application. More detailed explanation of those profiling tables will be discussed in Section 2.2.3. Note that our proposed technique is better in mobile/embedded environment since running applications are limited and the number of running applications is less than high-performance microprocessors because it reduces the optimal DVFS level calculation overhead. When the process (thread) is re-fetched to cores, DVFS controller looks up the profiling table in DVFS controller (not in OS) and adjusts voltage and frequency level of each core. Voltage and frequency level according to DVFS level is represented in Table 1.

How to determine optimal DVFS level and support profiling are explained in Section 2.2.2 and 2.2.3, respectively.

2.2.2 Determining optimal DVFS level considering both power consumption and performance

It is crucial to consider both energy consumption and performance though there should be trade-off between them. Thus, determining proper DVFS level is

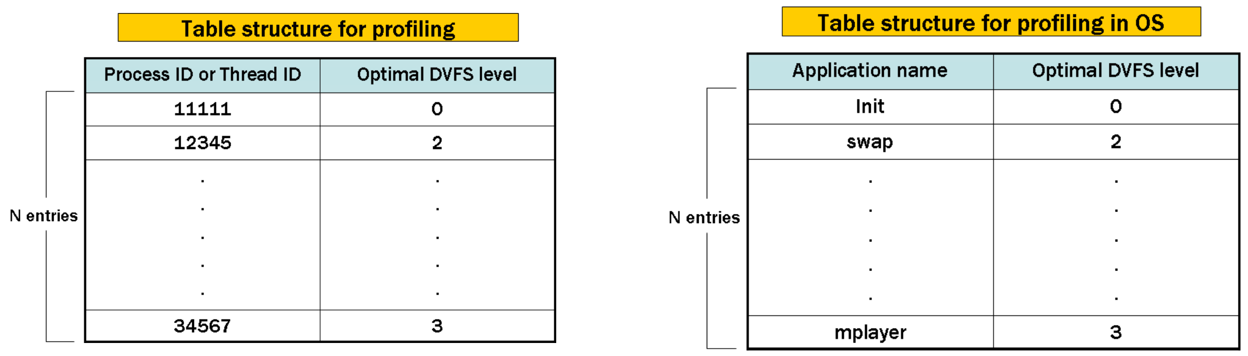
Table structure for profiling

N entries

Process ID or Thread ID	Optimal DVFS level
11111	0
12345	2
. . . . .	. . . . .
34567	3

Table structure for profiling in OS

N entries

Application name	Optimal DVFS level
Init	0
swap	2
. . . . .	. . . . .
mplayer	3

(a) Profiling table in DVFS controller **(b) Profiling table in DVFS controller**

Figure 2. Profiling support in DVFS controller and OS.

the most critical issue in our technique. To determine the optimal DVFS level for each core, we use Energy-Delay Product (EDP) as our main metric. Lower EDP value is better than higher EDP value. To calculate EDP value for each application, we extract energy and performance (execution time) information by executing applications. Energy consumption and performance can be determined by program behavior analyzer such as SimPoint [9]. (We will explain more about it in next subsection.) In this paper, we extracted energy consumption and execution time using SimPoint tool which can pick up the most representative simulation points (100 million instructions). Using program behavior analyzing tool, we do not need to run the entire applications from start to end.

As a result, EDP value is calculated by equation (1). Note that CPI (Clock cycle Per Instruction) x 10^8 x cycle time is the execution time in equation (1).

$$EDP = Energy\ \ consumption(J) \times CPI \times 10^8 \times cycle\ \ time \quad (1)$$

Additionally, ED^2P can be used for another metric which gives more weight to delay metric. ED^2P. Equation (2) describes how to calculate ED^2P value.

$$ED^2P = EDP \times CPI \times 10^8 \times cycle\ \ time \quad (2)$$

After calculating each EDP or ED^2P value for each DVFS level, optimal DVFS level which has the least EDP or ED^2P value is determined. That can be simply represented by equation (3). We assume that system has *n* DVFS levels. Note that calculating least ED^2P value is entirely same as equation (3) except using ED^2P instead of EDP.

$$Minimum\ \ EDP = \min(EDP_{level\ 0}, EDP_{level\ 1}, EDP_{level\ 2} \cdots EDP_{level\ n}) \quad (3)$$

Matching DVFS level which has minimum EDP value is optimal DVFS level for that application. Those values are calculated by operating systems (OS) and profiled in DVFS controller as well as in OS.

Table 2. Architectural parameters per one processor core

Processor Core	
Instr. Window	RUU 80, LSQ 64
Issue width	4 (Int: 4 FP: 2 LSQ: 2)
Functional Unit	4 Int ALU / 1 Int Mul/Div 2 FP ALU / 1 FP Mul/Div 2 Memport
Branch Prediction	
Local History Table	Combined, Bimodal 4K table
Local Predict	2-Level (GAg) 1 table,
Global History Register	12bit history
Choice Predict	4K chooser
BTB	2K entry, 2-way
Return-address stack	32 entry
Memory Hierarchy	
L1 D-cache Size	64K, 2-way (LRU)
L1 D-cache Assoc.	64B blocks, 2cycle latency
L1 I-cache Size	64K, 2-way (LRU)
L1 I-cache Assoc.	64B blocks, 2cycle latency
TLB Size (full assoc.)	128 entry fully assoc, 4K pages 30 cycle miss latency

2.2.3 Profiling support in DVFS controller

Calculating DVFS level whenever the application is fetched is too costly. Thus, we use small data structure for profiling DVFS level for each application. It can be implemented by using small table. Figure 2 depicts simple table structure for profiling support.

Figure 2 (a) is the table structure in the DVFS controller. The first column is process ID or thread ID which represents the application. The second column is determined DVFS level according to process ID or thread ID. In order to offer design flexibility, we do not determine the number of entries. The number of entries can be determined considering the number of processes or threads running in the system.

When the application is executed at first, there is no energy consumption and execution time information. Consequently, we cannot determine proper DVFS level. Thus, we can use program behavior analyzing tool such as SimPoint [9] as we explained in previous subsection. The newly executed application is executed using analyzing tool and proper DVFS level information is profiled in operating systems as well as in DVFS controller. Figure 2 (b) depicts profiling table in OS. Since proper DVFS level of once executed application is profiled in OS, we can reduce DVFS level calculation overhead significantly. The name contention of applications may be the problem in this table. We can simply resolve this problem by numbering or other detecting methods.

3. Evaluation

3.1 Evaluation Methodology

To evaluate our proposed technique, we used SimpleScalar [3] /Wattch [2] toolset to evaluate performance and power. We used Alpha 21364 processor parameters. Table 2 shows architectural parameters in one core. Each core has the same configuration. We choose 8 applications (crafty, gzip, gcc, mcf, twolf, applu, mesa, fma3d) from SPEC2000 benchmarks. Selected 8 applications are the most representative applications among 26 SPEC2000 benchmarks [6]. Context-switching time slice is 50ms which reflects the time slice of the modern operating systems [10]. The process technology is 100nm technology.

Table 3. EDP values for each application

EDP (J x s)	gzip	gcc	crafty	mcf	twolf	applu	fma3d	mesa
DVFS Level 0	996595	1550060	970838	26039231	2322011	1899524	1195289	628152
DVFS Level 1	801201	1388160	782435	37776249	1865234	2964043	1520246	531906
DVFS Level 2	805181	1600144	790579	73468766	1878848	5758059	2495636	569490
DVFS Level 3	854167	1879967	842490	120178198	1997045	9320445	3690629	633789

Table 4. ED^{2}P values for each application

ED^{2}P (J x s^2)	gzip	gcc	crafty	mcf	twolf	applu	fma3d	mesa
DVFS Level 0	76641	139835	73845	7501870	257287	194089	102484	40625
DVFS Level 1	30977	66061	30011	6884800	104198	201859	81060	17648
DVFS Level 2	20902	54155	20428	11525590	70720	345793	110756	13002
DVFS Level 3	16726	50068	16464	16687760	56842	500634	142127	11119

Table 5. Optimal DVFS levels for each application

	gzip	gcc	crafty	mcf	twolf	applu	fma3d	mesa
EDP	1	1	1	0	1	0	0	1
ED^{2}P	3	3	3	2	3	0	1	3

3.2 Optimal DVFS levels

First, we extract EDP and ED^{2}P values for 8 applications. Using SimPoint [9] tool, we extract the most representative 100 million instructions which reflects the most exact program behavior. Table 3 and 4 shows EDP and ED^{2}P values for each application.

As shown in Table 3, each application has different EDP values due to the difference of program behavior. Mcf which is the most memory-bound application shows the highest EDP values because the execution time is significantly higher than any other applications. Other applications show various EDP results which depend on the program behavior. As described in Table 4, ED^{2}P values show a little difference compared with EDP results. Like EDP, mcf application shows the highest ED^{2}P value.

Lastly, we determine the optimal DVFS level for each application. Table 5 shows the optimal DVFS level. It shows higher DVFS level when considering ED^{2}P than considering EDP. For this reason, ED^{2}P gives more weight to execution time. Thus, it makes cores run in high frequencies which can reduce the execution time.

3.3 Energy-Delay efficiency

3.3.1 Analytical Model for evaluating Energy-Delay efficiency

We propose analytical model for evaluation. To calculate EDP or ED^{2}P in multi-core system, we should calculate EDP or ED^{2}P and normalize it to the baseline scheme (It will be discussed in next subsection.) Lastly, we calculate the average EDP or ED^{2}P values of each application running at each core.

First, we calculate per core EDP value. Equation (4) represents EDP in one core (core i).

$$EDP_{core\,i} = Energy\ consumption_{core\,i} \times (execution\,time + context\ switching\ overhead)_{core\,i} \quad (4)$$

The next step is to calculate normalized average of each core. The reason why we calculate normalized average is that if one core shows a large EDP value, the result can be biased to a large EDP value. Thus, we use normalized average of each core. Equation (5) denotes normalized average of each core.

$$Normalized\,EDP_{core\,i} = \frac{EDP_{core\,i}\ of\ the\ proposed\,technique}{Baseline\,EDP_{per\,core\,i}} \quad (5)$$

The last step is to calculate geometric mean of per core normalized EDP values. To calculate normalized per core ED^{2}P value, we simply replace EDP value to ED^{2}P value when calculating normalized per core ED^{2}P values.

3.3.2 Energy-Delay efficiency of our proposed technique

We evaluated our technique compared to the performance-biased scheme (all of the cores run at DVFS level 3.) and the energy-biased scheme (all of the cores run at DVFS level 0). We assume that two applications are assigned to one core. Table 6 represents the assignment of applications to each core. Note that voltage and frequency of our simulated processor cores are not statically determined but freely adjust DVFS level from 0 to 3. In other words, which

Table 6. Application assignment of each core

Core 0	Core 1	Core 2	Core 3
gzip	gcc	crafty	mcf
twolf	applu	fma3d	mesa

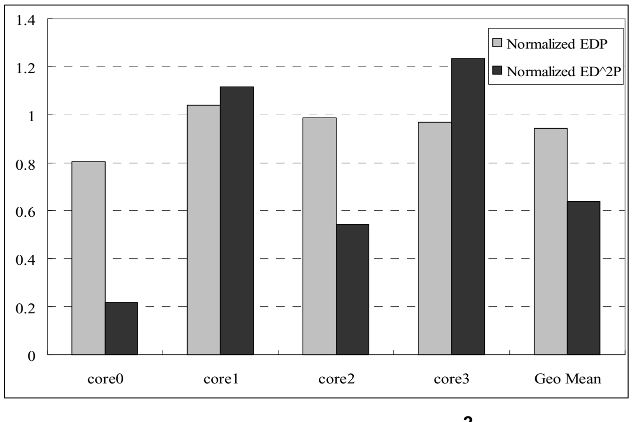

Figure 3. Normalized EDP and ED^{2}P compared to energy-biased scheme

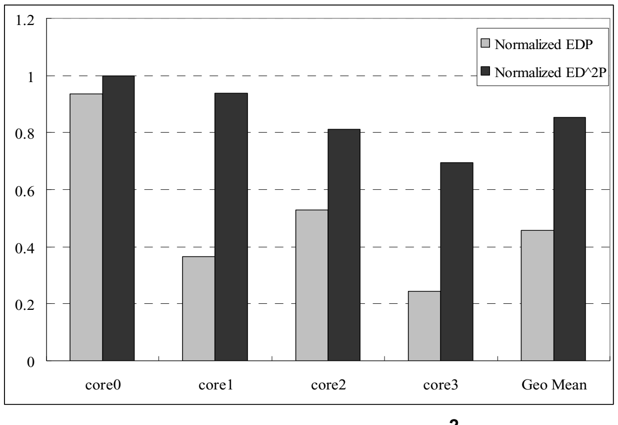

Figure 4. Normalized EDP and ED^{2}P compared to performance-biased scheme

application is assigned to a specific core is not critical factor because all of the cores are identical (Each core can freely adjust their DVFS levels.). Context switching overhead is 0.5us [7].

Figure 3 and 4 depict our evaluation results compared to energy-biased scheme and performance-biased scheme, respectively. As shown in Figure 3 and 4, the proposed technique shows high energy-delay efficiency. Compared to energy-biased scheme, our proposed technique reduces 5.6% of EDP and 36.4% of ED^2P, on average. However, in case of core 1 and 3, EDP or ED^2P are increased a little. For this reason, we calculate our optimal DVFS levels of each application not considering multi-programming environment. Since two applications are executed in one core, there can be a little difference results according to the assignment of applications to each core. Assignment of applications is out of scope in our paper though it can also be an interesting issue related to our proposed technique. However, there are significant EDP and ED^2P reduction in terms of overall systems. Compared to performance-biased scheme, our proposed technique reduces 54.3% of EDP and 14.7% of ED^2P on average. Through trade-off between energy and delay, our proposed technique shows no-biasing effect to either energy or performance.

4. Conclusions

In this paper, we propose a low-cost DVFS technique for multi-core processors. Considering both energy and delay of the applications, the proposed technique reduces EDP and ED^2P significantly as well as effectively. The proposed technique shows 5.6% and 54.3% EDP reduction compared to the energy-biased and performance-biased scheme, respectively. In case of ED^2P, the result shows 36.4% and 14.7% reduction compared to the energy-biased and performance-biased scheme, respectively. It is because our proposed technique is not biased either energy or performance.

As process technology advances, the energy/power problem becomes more and more severe. The energy/power problem is also important in multi-core system as well as uni-processor system. Furthermore, severe energy consumption becomes serious wall in mobile devices. However, performance is also important metric for microprocessors. Though a technique is energy-efficient, it is meaningless if performance degradation is severe. The proposed technique can be good alternative in future multi-core system where both energy consumption and performance are critical.

Acknowledgement

This work was supported by the Second Brain Korea 21 Project and Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R01-2007-000-20750-0).

Reference

[1] S. Borkar. “Design Challenges of Technology Scaling”. IEEE Micro, 1999.

[2] D. Brooks, V. Tiwari, and M. Martonosi. “Wattch: a fra mework for architectural level power analysis and optim izations”. In proceedings of HPCA 2000. Feb 2000.

[3] D. Burger, A. Kagi, and M. S. Hrishikesh. “Memory hie rarchy extensions to simplescalar 3.0”. Technical Report TR99-25, Department of Computer Sciences, The Univ ersity of Texas at Austin, 2000.

[4] J. Donald, M. Martonosi. “Techniques for Multicore Th ermal Management: Classification and New Exploratio n.” In proceedings of ISCA 2006. June 2006.

[5] J. Hasan, A. Jalote, T. N. Vijaykumar, C. E. Brodley. “H eat Stroke: Power-Density-Based Denial of Service in S MT.” In proceedings of HPCA 2005. Feb 2005.

[6] X. Liang, R. Canal, G. –Y. Wei and D. Brooks. “Process Variation Tolerant 3T1D-Based Cache Architectures”. In proceedings of MICRO ’07. December 2007.

[7] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri. “How low can you go? Recommendations for hardware-supported minimal TCB code execution.” In Proceedings of ASPLOS 08. Mar 2008.

[8] S. Palacharla, N. Jouppi, and J. E. Smith. “Complexity-Effective Superscalar Processors.” In proceedings of ISCA 97. June 1997.

[9] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. “Automatically characterizing large scale program beha vior”. In proceedings of ASPLOS 02. 2002.

[10] R. S. Shindi and S. Cooper. Evaluate the Performance C hanges of Processor Simulator Benchmarks When Cont ext Switches are Incorporated. In proceedings of ACM SIGAda’06, Nov 2006.

[11] ARM11 MPCore. Available at “http://www.arm.com/pr oducts/CPUs/ARM11MPCoreMultiprocessor.html”

[12] "Enhanced Intel SpeedStep Technology for the Intel Pen tium M Processor” Intel white paper. March 2004.