Empirical Analysis of Power Management Schemes for Multi-core Smartphones

Sangwook Kim
College of Info. & Comm.
Sungkyunkwan University
Suwon, South Korea
swkim@csl.skku.edu

Hwanju Kim
Dept. of CS
Korea Advanced Institute of Science and Technology
Daejeon, South Korea
hjukim@calab.kaist.ac.kr

Jongwon Kim
Telecomm. Department
Samsung Electronics Co.
Suwon, South Korea
jz.kim@samsung.com

Joonwon Lee
College of Info. & Comm.
Sungkyunkwan University
Suwon, South Korea
joonwon@skku.edu

Euiseong Seo
College of Info. & Comm.
Sungkyunkwan University
Suwon, South Korea
euiseong@skku.edu

ABSTRACT

Dynamic power management schemes in mobile devices such as smartphones and tablet PCs enhance battery life at the cost of prolonged user-perceived response time while the response time is a crucial factor for user experience. This paper presents systematic analysis of existing power management schemes adopted in recent smartphones in terms of user-perceived response time and energy consumption. For this analysis, we developed a latency measurement benchmark tool to quantify responsiveness to user inputs and used it with an externally-connected power meter to concurrently measure energy consumption and response latency. The evaluation showed that some existing DVFS schemes can significantly harm the response time. More seriously, the analysis revealed that the processor hotplug technique for multi-core systems may reduce responsiveness even without any gain in energy savings.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement Techniques

General Terms

Performance

Keywords

power management, multi-core, smartphones, DVFS, measurement, benchmarks

1. INTRODUCTION

As smartphones and tablet PCs are replacing conventional PCs, gaming devices and many other consumer electronics, their applications are requiring increasingly powerful performance. In order to meet such demand, most vendors continuously enhance their computing components such as application processors (APs) and graphics processing units (GPUs). With regard to APs, unlike wimpy microprocessors in traditional embedded systems, clock frequency and hardware parallelism become comparable to the desktop counterparts. Indeed, quad-core processors are already common in cutting-edge smartphones and tablets. Furthermore, the clock frequency of commercial APs recently hit 2.5GHz [1].

High performance computing components in mobile devices, however, brought up an issue on battery life, which is one of the most crucial concerns mobile users commonly have. Although multi-core processors are meant to be energy-efficient units, high clock frequency and increasing number of cores could drain limited battery more rapidly than low-power single-core embedded processors. To address this problem, most off-the-self smartphones and tablets adopt power management schemes that make use of *dynamic voltage and frequency scaling* (DVFS) [7, 15] and *processor hotplugging* [8]. Considering that energy saving and high performance are conflicting goals, many researchers have studied to optimize power consumption of processors while maintaining desired performance levels [11, 5, 12].

The existing power management schemes being widely used in mobile and desktop systems are typically implemented at operating system (OS) layer in order to be deployed without burdening to users or developers. For example, Linux provides several kernel-level DVFS policies on which Android-based mobile devices rely to manage power consumption. To effectively deal with the trade-off between power and performance, power management schemes should estimate the performance demand of an application with which a user interacts. On the basis of estimated performance demands, the OS can adjust clock frequency, or turn on and off cores on the fly. Most OS-level power management schemes regard CPU load as an indicator of performance demand, since the workload performance is generally associated with the CPU load.

However, the existing power management algorithms us-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICUIMC(IMCOM)'13 January 17-19, 2013, Kota Kinabalu, Malaysia.
Copyright 2013 ACM 978-1-4503-1958-4 ...$15.00.

ing CPU load as their performance decision basis tend to focus on throughput rather than responsiveness. Such schemes may hurt responsiveness, since a decision on adjustment of processor performance is reactively made after CPU load increase is observed. In this regard, agility of power management in response to CPU load increase is a major factor to provide sufficient user responsiveness. Although load-based power management schemes are widely deployed, there has been no work on quantitative analysis of various load-based DVFS and processor hotplug policies from the perspective of user responsiveness and energy consumption.

This paper presents empirical analysis of various OS-level power management schemes, DVFS and processor hotplug policies, which are used by real-world multi-core smartphones. The analysis is aimed to figure out the extent to which each power management scheme affects user responsiveness and how much energy saving is finally achieved by each scheme. In order to accurately quantify user-perceived latency, we developed a latency measurement benchmark, named *LatencyBench*, which measures the time elapsed between a user input triggered and browser launch completed. For power measurement, we used a power meter externally connected to the evaluated device. With this measurement environment, we examined five Linux DVFS policies, which consist of two static and three dynamic policies, and a load-based processor hotplug scheme in an Android-based device.

Our findings obtained from this analysis are summarized as follows: First, lazy (or conservative) ramping up of clock frequency in response to CPU load increase leads to degradation of responsiveness that in turn affects user perception (by hundreds of milliseconds compared to agile ramp-up) without meaningful gain in energy savings. Second, user responsiveness under agile ramp-up DVFS policies is close to the performance achievement at the highest clock frequency. Third, a load-based processor hotplug policy shows similar user response time even to the single-core case due to the high latency of bringing an additional core online after detection of load changes. Through the analysis with various parameter settings for load monitoring, we found that processor hotplug latency itself intrinsically prolongs response time. Finally, dual-core configuration without hotplugging contributes both to user responsiveness and energy efficiency by inducing short active working cycles and long idle period.

The rest of this paper is organized as follows: Section 2 introduces existing power management schemes for smartphones, and the analysis environment are presented in Section 3. Section 4 evaluates and analyzes the power management schemes with the proposed measurement technique, and Section 5 concludes this research.

2. BACKGROUND

This section covers the details regarding the power management schemes in smartphones. First, the OS-level processor power management policies in the Linux kernel, which is used as the OS kernel for many existing smartphone OSes, are introduced. Next, an OS-level multi-core power management scheme is explained.

2.1 DVFS Management

Most modern smartphones are equipped with DVFS-enabled APs. The implementation details such as the maximum and minimum frequencies vary among different processors. In order to cope with the underlying implementation differences, Linux abstracts the DVFS control of processors, and then exposes a simple interface to the upper layer. Using this interface, a kernel module called *governor* controls the performance level of processors. Generally, there are two kinds of governors; the static and dynamic governors.

The static governors simply maintain a predefined performance level regardless of the system status. For example, the *performance governor* keeps the clock frequency to the highest supported value, whereas the frequency is statically set to the lowest one under the *powersave governor*.

On the other hand, the dynamic governors adjust the processor clock speed on-the-fly according to the performance demand of the system. For instance, the *ondemand governor* [9] increases the clock speed to the highest level when the system load is above the predefined threshold by periodically monitoring the system load on the processor. On the contrary, the ondemand governor decreases the clock speed when the system load becomes below the predefined threshold. The *conservative governor* [9] is the modified version of the ondemand governor, which adjusts the performance level conservatively. Unlike the ondemand governor, the conservative governor increases the clock frequency gradually when the system load rises above the predefined threshold, and decreases the frequency step by step in response to sufficiently low system load.

The two dynamic governors mentioned above depend on periodic monitoring of system load, which could lead to delayed response to sporadic user inputs. The *interactive governor* [3], as its name suggests, is designed for interactive workloads requiring fast reaction. Basically, the interactive governor uses periodic load-based technique similar to other governors. However, the interactive governor delegates the procedure for adjusting the clock frequency to a kernel thread with the highest real-time priority for preventing delayed reaction due to scheduling contention with other tasks. In addition, this governor adjusts the clock frequency when a processor comes out of idle state. Hence, the interactive governor can quickly increase the clock frequency in response to the sporadic user-generated events such as screen touches or button presses.

Most Linux kernel-based smartphones currently are using one of the dynamic governors mentioned above or a variant of them. Especially, the ondemand governor is used as the default governor by most Linux kernel-based smartphones. Recently, the interactive governor is adopted as the default governor for the official Android platform.

2.2 Multi-core Management

Increasing the number of processor cores incurs significant power consumption increase. To mitigate this increased power consumption, *processor hotplugging* [8] is widely used for smartphones. By using processor hotplugging, unnecessary cores remain in the lowest power state until they become required again. Unnecessary cores can be identified using the system statistics similar to that of the dynamic governors.

The representative processor hotplugging algorithm, which is applied to the commercial dual-core smartphone, is presented in Figure 1. Suppose that the target processor has dual cores. This algorithm periodically monitors average load of on-line cores, and then turns off the second core (Core-1) if either the average load is less than the low-threshold value or the processor clock has been adjusted

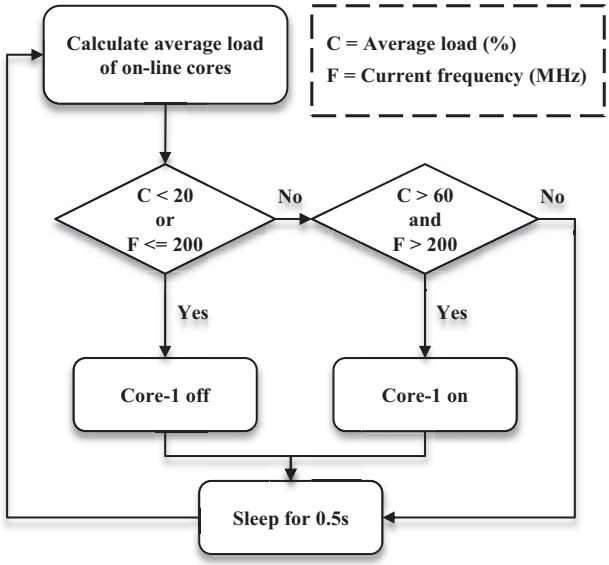

Figure 1: The processor hotplug algorithm in the commercial dual-core smartphone. (200 Mhz is the lowest supported clock speed in the target platform.)

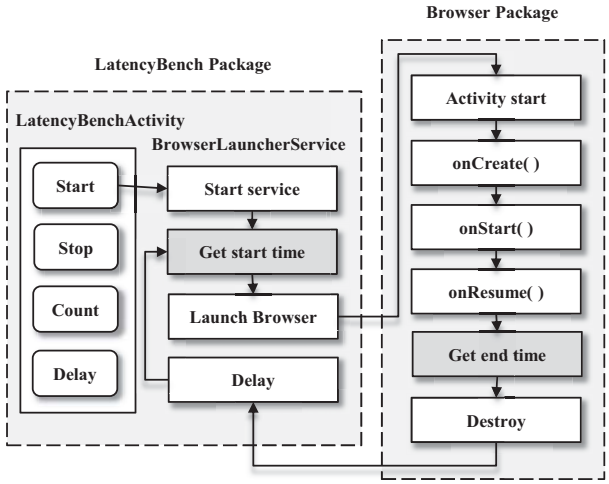

Figure 2: Execution flow of LatencyBench.

to the lowest value by the governor. Conversely, Core-1 is turned on when the average load rises above the high-threshold value and the governor has adjusted clock frequency to a greater value than the lowest one. Although the implementation details may differ depending on APs, the introduced approach is used as the fundamental ppolicy in the currently available hotplug implementation [2].

Though the processor hotplugging technique can reduce power consumption, it can also cause delayed response to user inputs because turning on a core inherently incurs both hardware and software latency [10]. To quantify the aggregated latency of processor hotplugging in real systems, the time from initiation of the activation procedure to its completion was measured on the dual-core embedded board. (The detailed information about the board is given in Section 3.) According to this measurement, the activation procedure was revealed to take approximately 100 ms to bring the second core to on-line. The aggregated latency includes time spent in both OS and hardware layers. Since this latency is not negligible, the parameter values that control the processor hotplugging algorithm to determine the monitoring period should be carefully selected to minimize unnecessary power consumption while fully utilizing the computation power of a multi-core processor.

3. ANALYSIS METHODOLOGY

This section addresses the design and implementation of LatencyBench, which measures the response delay of systems to assess the quality of user experience. In addition, this introduces both hardware and software configurations of the experimental system.

3.1 LatencyBench

To assess the power management schemes in terms of user experience, measurement of response time under each power management scheme is quintessential. However, most existing performance measurement tools or benchmark suites for smartphones are designed only to measure throughput of the systems.

In order to fill this gap, we implemented a response delay benchmark tool, LatencyBench. LatencyBench measures the time interval between invocation of an application by user inputs and completion of the application launch process. LatencyBench does not require any modification of the existing systems since it is implemented as a user-level application.

LatencyBench is a user-level application written in Java. It consists of two packages; the LatencyBench package and the Browser package as shown in Figure 2. A thread in the LatencyBench package, which was named LatencyBenchActivity, obtains both number of evaluation and time delay between every run from users, and then periodically instantiates the browser launching service according to the given parameters. The browser launching service, BrowserLauncherService, records the time of every instantiation start and invokes the web browser through the Browser package. The Browser package is a modified version of the Android default web browser. It records a time stamp when it finishes initialization and gets ready to operate. After marking the time stamp, it destroys itself and the context returns to BrowserLauncherService. LatencyBench considers the elapsed time between these two time stamps as response delay of the web browser launch activities. This evaluation loop is repeatedly conducted as many times as the given number of iteration. In order to stabilize clock frequency according to a policy of the applied governor, LatencyBenchActivity pauses for the given time delay between every iteration.

3.2 Experimental Hardware

As an evaluation platform, this research employed an embedded board for commercial tablets and smartphones. The board is equipped with an Exynos 4210, a dual-core AP operating at 1 Ghz, and the main memory size is 1 GB. In every experiment, the display brightness of the board was set to 50% of its maximum. As an OS, Android 2.3.4 was installed on top of the embedded board.

Figure 3: Experimental setup.

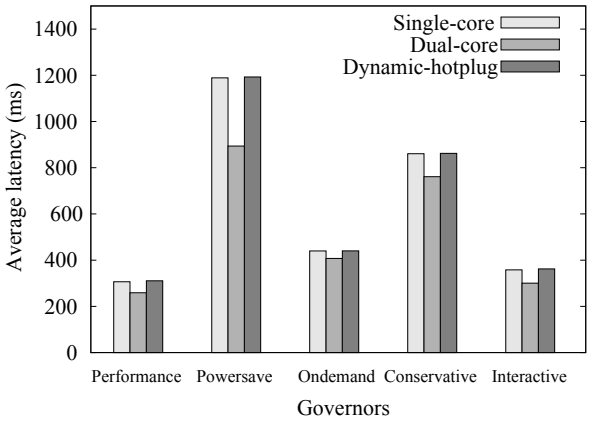

Figure 4: Average response time of various power management policies using LatencyBench.

The power consumption was measured with a Yokogawa WT-210, a digital power meter. The power meter is connected to the D/C power input to the embedded board. The power meter includes a data acquisition unit and the unit is attached to a PC. The data produced by the power meter are collected by the PC through the data acquisition unit. The sample rate of data is 100 ms, and the accuracy of measurement is over 99.9% according to its specification sheet. Figure 3 shows the experimental environment.

In order to compare the various power management schemes, several Linux power management governors were incorporated into the experimental platform. In addition, to assess the effectiveness of multi-core management schemes, the experimental system was configured with three multi-core management policies. The first policy is *Dynamic-hotplug*. It is a variant of the processor hotplug approaches, which were introduced in Section 2. Second, the system was set to use only one of the two cores. This is dubbed as *Single-core*. In the third scheme, *Dual-core*, both cores are always actively used.

4. EVALUATION

This section presents the experimental results and analysis of the DVFS governors. Then, the experiments of Dynamic-hotplug with various configurable parameters follow to analyze its effectiveness in detail.

4.1 Experimental Results

To measure the performance in terms of response time, ten consecutive launches of the Browser are carried out using LatencyBench, and then the recorded latencies are averaged for comparison. Note that a total execution time of the benchmark is different depending on the applied power management schemes. Hence, in the measurements of energy consumption, energy consumption in idle period with display-on is padded to energy consumption during execution of the benchmark on the basis of execution time of the slowest one, to fairly compare the management schemes. The results with an assumption of display-off idle period are omitted because they showed similar trend with the display-on case.

Figure 4 shows the average response latency of various combinations of DVFS and multicore management schemes using LatencyBench, and Figure 5 shows the energy consumption. As expected, the performance governor achieved better responsiveness than the powersave governor whereas the performance governor consumed more energy than the powersave governor.

The system responded more quickly when using the ondemand governor than the conservative governor. This result stems from the difference between the algorithm policies about the conditions to increase the clock frequency. The conservative governor increases the clock frequency gradually in response to high load such as launching a web browser whereas the ondemand governor promptly increases the clock speed to the highest value when the system load hits the ceiling. Note that there was only little difference in energy consumption though the difference in response time was significant.

The interactive governor achieved better responsiveness than the ondemand governor since the interactive governor instantly activates its DVFS algorithm and raises the clock speed when a processor comes out of idle state. This aggressive adjustment caused more energy consumption in comparison to ondemand as shown in Figure 5. However, the difference of energy consumption between the ondemand and the interactive governor is insignificant compared to the gain in the reduced response time.

According to these results, there were significant differences in response time depending on the applied governors. In terms of energy consumption, however, only little differences were observed across various governors. This result can be explained through the fact that the processor used in the experiments consumes only little portion of the overall power consumption in comparison to powerful processors being used in servers or PCs [4, 13]. In addition, the length of the idle period affects the overall energy consumption. For example, the ondemand governor completes its execution of the benchmark earlier than the conservative governor. Thus, the processor consumes only the idle power for the rest of the time. This gap in the power consumption during the idle period actually reduces the overall energy consumption difference between the ondemand and conservative governors in spite of the relatively long period of high frequency under

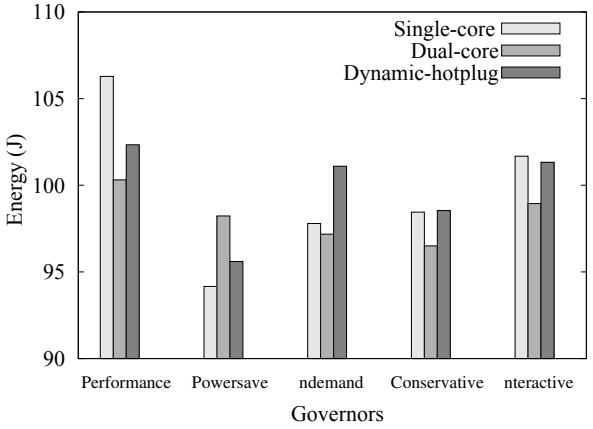

Figure 5: Energy consumption of various power management policies during the execution of LatencyBench. (display is on during idle period.)

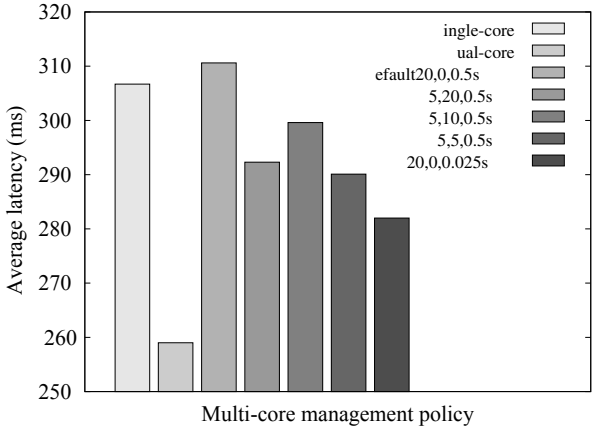

Figure 6: Average response time under various multi-core management policies. (x %, y %, z s) = (low threshold to turn off, high threshold to turn on, monitoring period.)

Table 1: Average wait time of the Browser process in the run-queue under different multi-core management policies with the performance governor

Management Policy	Average Wait Time (us)
Single-core	907
Dual-core	282
Dynamic-hotplug	903

ondemand.

Regarding the core management policies, Dual-core showed lower latency than Single-core in all combinations with the DVFS governors. Interestingly, the response time under Dynamic-hotplug was close to that under Single-core though both cores are supposed to be in activation state as system load increases under Dynamic-hotplug. This result implies that Dynamic-hotplug cannot effectively utilize the computation power of the multi-core processors. In addition, Dynamic-hotplug consumed more energy than Dual-core in most cases because it completed the benchmark slower than the Dual-core cases. Similarly, Single-core consumed more energy than Dual-core in most cases due to the extended execution time, and thus the shortened idle time.

In order to further analyze the results on the multi-core management policies, scheduling delay during the experiments was measured. Table 1 shows the average wait time of the Browser process in the run-queue when the three management policies are applied in combination with the performance governor. The average wait time of the Dual-core case was significantly shorter than that under both Single-core and Dynamic-hotplug. In addition, the average wait time of Dynamic-hotplug was very similar to Single-core, which again means that Dynamic-hotplug could not take advantage of the second core. These results suggest that the Dynamic-hotplug algorithm, with its default parameters, fails to rapidly react to instant high load, which is commonly found in smartphone workloads.

To explore the possible improvement of hotplugging effectiveness, additional experiments were conducted by varying values of three configurable parameters of the Dynamic-hotplug algorithm. The configurable parameters are the high-load threshold to turn on the second core, low-load threshold to turn off the second core, and monitoring period that determines the granularity of the decision cycle.

Figure 6 shows the average response delay according to various configurations of the Dynamic-hotplug policy. The average latency of Dynamic-hotplug was reduced compared to its default configuration when both high-load and low-load threshold values were lowered, because lowering the thresholds made the algorithm more sensitive to the system load. Moreover, the average response time was further reduced than the other configurations when the monitoring period was shortened to 0.025 s with the other parameter values same as the default setting. This result implies that the monitoring period is more critical to the user-perceived response time than the threshold values.

However, Dynamic-hotplug with various parameters could not perform as fast as Dual-core. This limitation arises from the inherent delays of processor hotplug. Generally, Dynamic-hotplug triggers activation of the second core when it senses that the current load hits the high threshold. However, the core activation time, which is approximately up to 100 ms as described in Section 2, prohibits scheduling of the newly created heavy load processes on the second core. This scheduling delay cannot be eliminated by the customization of the parameter values.

Figure 7 shows energy consumption under various configurations of the multi-core management policies with the performance governor. The energy consumption under the Dual-core setup was lower than that under any other policies because Dual-core finished the workload in the shortest time and stayed in the longest idle state. The energy consumption under Dynamic-hotplug was generally reduced as the parameters were changed to the aggressive values. The amount of energy consumed under Dynamic-hotplug with the default parameters is lower than that in the cases where the two threshold values were adjusted to increase sensitivity for changing system load. With the default configuration, Dynamic-hotplug reacted tardily to the load changes, and in consequence, the second core became online in a few milliseconds after the Browser process began to execute.

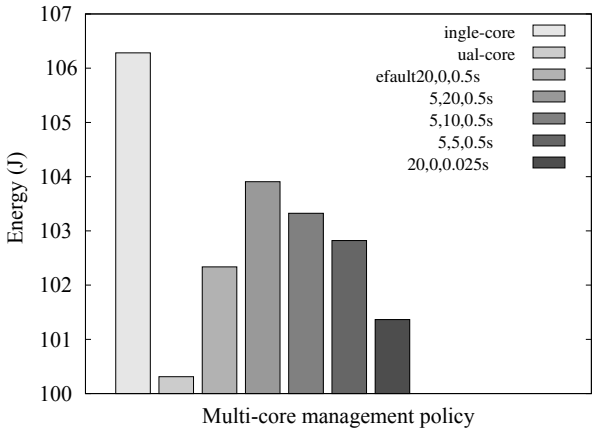

Figure 7: Energy consumption under various multi-core management policies. (x %, y %, z s) = (low threshold to turn off, high threshold to turn on, monitoring period.)

Note that the differences of energy consumption among the core management policies are insignificant in comparison to the differences in average response latency. The maximum gap of response latency between the best case and the worst case was 20 % while that of energy consumption was only 6 %.

4.2 Analysis and Implications

According to the results, user-perceived response time is highly dependent on which power management scheme is used whereas the effect to the energy efficiency is relatively insignificant. Conservative DVFS policies remarkably degrade user-perceived responsiveness compared to that of aggressive DVFS policies. Though aggressive DVFS policies typically consume more energy than conservative policies, the differences in energy consumption among various policies were negligible.

These results stemmed from the power consumption constitution in modern mobile systems. According to the previous research, processor power consumption takes little proportion in overall power consumption of a mobile system [13]. Moreover, similar to desktops or servers [14], a processor for a smartphone is expected to integrate more transistors in a single core, and both the size of SRAM and the number of cores per processor also increasing. As a consequence, static power of a processor is continuously increasing whereas dynamic power is decreasing. This trend implies that energy savings by using the DVFS are expected to be reduced since DVFS can reduce only dynamic power consumption. Thus, the margin in power consumption that can be saved by DVFS management schemes is low in modern APs.

However, the impact of the power management schemes on user-perceived responsiveness are direct. Considering the trend on mobile processors and importance of responsiveness in smartphones, response time should be given higher priority than energy consumption when designing processor power management schemes for smartphones.

In the case of multi-core management schemes, Dynamic-hotplug showed pathological behaviors during the experiments due to the unacceptable hotplug delay and its inappropriate parameter settings. In some cases, dynamic multi-core management leaded to even more energy consumption than the case that the dynamic management policy was not applied (i.e., Dual-core). Though the system's responsiveness is sacrificed for possible energy savings, the carelessly designed algorithm did not work successfully. This is contradictory to the purpose of dynamic core state management.

Therefore, dynamic multi-core management policy should carefully decide when to activate or deactivate reflecting the inevitably incurred hotplug delay. In addition, considering processor improvement on energy-efficient idle states, it is highly encouraged to use advanced idling states whenever possible as an alternative to dynamic hotplug techniques.

The ideal goal of a power management scheme is to use high clock frequency only in performance-critical interval. To accomplish this goal, the existing power management schemes typically compare the current system load to predefined threshold values to determine the performance-critical time interval. However, system load was identifed not to be an effective marker for performance demand because there are CPU-intensive background tasks which have little impact on responsiveness for a smartphone user.

In order to deal with the inherent limitation of OS-level approach, some researchers have investigated possibilities of user-driven processor power management schemes. The principle of user-driven techniques is to exploit user-perceived latency as a metric for dealing with the performance-energy trade-off. For example, Zhong et al. [16] used prediction mechanism based on human-computer interaction history and theories from the filed of psychology in order to decide adequate performance level of a processor. On the other hand, Mallik et al. [6] proposed a user-driven frequency scaling scheme (UDFS) which dynamically changes clock frequency through a direct user feedback mechanism.

However, none of the proposed techniques are designed for multi-core smartphones, which have different characteristics from laptops or cellphones. Generally, there is only one foreground task, which interacts directly with a user, at a time with some background tasks in a smartphone. Hence, it would be better to exploit application or framework-level information such as application launches or screen touches because they can directly recognize the start and end points of user interaction without requiring complex prediction mechanisms or unnecessary user involvements.

5. CONCLUSION

This paper analyzed the effectiveness of various power management schemes for multi-core smartphone systems in terms of energy efficiency and user-perceived response latency. To achieve this goal, this paper presents Latency-Bench, which is a simple benchmark tool that measures the response delay of smartphone systems.

The evaluation showed that the existing power management schemes have different characteristics in aspects of energy efficiency and user-perceived responsiveness. Specifically, the experimental results showed that the use of a simple processor hotplug algorithm may harm the responsiveness of systems because it does not timely react to spontaneous user inputs and, in turn, cannot fully utilize all cores.

An OS-level power management has inherent limitation for identifying performance-critical interval because of insufficient information. In order to deal with this limitation, we are working on a user-driven processor power management

scheme for multi-core smartphones as our future work.

In addition, asymmetric multi-core processors are expected to be widely used in smartphones and other consumer electronics in the near future. Since they have totally different performance and power characteristics in comparison to the conventional symmetric multi-core processors, we are also planning to investigate the issues around power management schemes for asymmetric multi-core processors in the follow-up research.

6. ACKNOWLEDGEMENTS

This research was supported by Basic Science Research Program (2010-0003453) and Next-Generation Information Computing Development Program (2012-0006423) through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology.

7. REFERENCES

[1] ARM A15 MPCore. `http://www.arm.com/products/processors/cortex-a/cortex-a15.php`.

[2] Hotplug governor. `https://wiki.linaro.org/WorkingGroups/PowerManagement/Doc/Hotplug`.

[3] Interactive governor. `https://lkml.org/lkml/2012/2/7/483`.

[4] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *In Proceedings of the 2010 USENIX Annual Technical Conference*, 2010.

[5] J. R. Lorch and A. J. Smith. Operating system modifications for task-based speed and voltage scheduling. In *In Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, 2003.

[6] A. Mallik, B. Lin, G. Memik, P. Dinda, and R. P. Dick. User-driven frequency scaling. *Computer Architecture Letters*, 5(2):1–4, Dec. 2006.

[7] T. Mudge. Power: A first class architectural design constraint. *IEEE Computers*, 34(4):52–58, Apr. 2001.

[8] Z. Mwaikambo, A. Raj, R. Russel, and J. Schopp. Linux kernel hotplug CPU support. In *In Proceedings of the Ottawa Linux Symposium*, 2004.

[9] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *In Proceedings of the Ottawa Linux Symposium*, 2006.

[10] S. Panneerselvam and M. M. Swift. Chameleon: Operating system support for dynamic processors. In *In Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012.

[11] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *In Proceedings of the 18th ACM Symposium on Operating Systems Principles*, 2001.

[12] E. Seo, S. Park, J. Kim, and J. Lee. TSB: A DVS algorithm with quick response for general purpose operating systems. *Journal of Systems Architecture*, 54(1–2):1–14, Jan. 2008.

[13] Y. Seo, J. Kim, and E. Seo. Effectiveness analysis of DVFS and DPM in mobile devices. *Journal of Computer Science and Technology*, 27(4):781–790, July 2012.

[14] E. L. Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *In Proceedings of the 2010 Workshop on Power Aware Computing and Systems*, 2010.

[15] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *In Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, 1994.

[16] L. Zhong and N. K. Jha. Dynamic power optimization targeting user delays in interactive systems. *IEEE Transactions on Mobile Computing*, 5(11):1473–1488, Nov. 2006.