An Event-Driven Power Management Scheme for Mobile Consumer Electronics

Sangwook Kim, Hwanju Kim, Jeaho Hwang, Joonwon Lee, and Euiseong Seo

Abstract — *Dynamic processor power management based on periodic monitoring of processor load is being widely used to enhance battery life of mobile consumer electronics. The existing power management schemes, however, often lead to poor user-perceived responsiveness, which is a crucial factor for the quality of user experiences, by making inadequate or delayed decisions on the processor performance level. This paper presents a novel event-driven scheme of the processor power management to guarantee high responsiveness while minimizing ineffective energy consumption. The proposed scheme exploits the characteristic of an interactive event, which is triggered by a user input instead of a periodic timer interrupt. The prototype of the proposed event-driven scheme is implemented for an application launch event as an example. In order to quantify user-perceived responsiveness and energy consumption simultaneously, this paper introduces a latency measurement benchmark program. In the evaluation with the benchmark, the proposed power management scheme showed comparable responsiveness with higher energy savings up to 20 % than the existing dynamic schemes when the interactive workload is mixed with a CPU-intensive background task.* [1]

Index Terms — Power management, User experience, Mobile consumer electronics, DVFS

I. Introduction

As mobile consumer electronics such as smartphones and tablet PCs are replacing conventional PCs, gaming devices and many other consumer electronics, their applications are demanding increasingly high performance. In order to meet such demand, most vendors continuously enhance the performance of their computing components such as mobile CPUs and graphics processing units (GPUs). With regard to mobile CPUs, 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 tablet PCs.

[1] This work was partly supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2012-0000148), and by the IT R&D program of MKE/KEIT [10041244, Smart TV 2.0 Software Platform].

Sangwook Kim, Joonwon Lee, and Euiseong Seo are with College of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea (e-mail: swkim@csl.skku.edu, joonwon@skku.edu, euiseong@skku.edu).

Hwanju Kim and Jeaho Hwang are with Department of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Korea (e-mail: hjukim@calab.kaist.ac.kr, jhhwang@calab.kaist.ac.kr).

Contributed Paper
Manuscript received 01/01/13
Current version published 03/25/13
Electronic version published 03/25/13.

High performance computing components in mobile consumer electronics, however, have brought up a problem on battery life, which is one of the most crucial concerns mobile users commonly have. High clock frequency and increasing number of cores in modern mobile CPUs could drain limited battery more rapidly than low-frequency single-core processors. To address this problem, most mobile consumer electronics adopt power management schemes that make use of *dynamic voltage and frequency scaling* (DVFS) [1], [2] and *processor hotplug* [3]. DVFS enables dynamic control of voltage and frequency level of each core in a processor whereas processor hotplug supports dynamic enabling or disabling processor cores to save more energy when the performance demand of a system is low. Based on these fundamental power management features, many researchers have studied to reduce energy consumption of processors while maintaining desired performance levels [4]-[8].

From the perspective of interactive systems such as mobile consumer electronics, a power management scheme should ideally provide the highest level of performance during a user-interaction interval in order to guarantee a fast response to a sporadic user input. In addition, the processor's performance should be maintained to the lowest level until the next user-interaction interval starts in order to minimize ineffective energy consumption, which may be involved in background computation. Hence, accurately identifying a user-interaction interval is crucial for high quality of user experience as well as extended battery life in mobile consumer electronics.

The existing power management schemes being widely used in mobile systems are typically implemented at operating system (OS) layer in order to be deployed without burdens to users and application developers. Most OS-level schemes use processor load to infer user-interaction intervals based on the assumption that a user input causes high processor load greater than the predefined threshold. These load-based schemes, however, would make false positive on a user-interaction interval when a CPU-intensive background task, which is irrelevant to user-interactions, exists in the system. Moreover, the load-based schemes may hurt responsiveness since a decision on adjustment of processor performance is reactively made after processor load increase is observed. As a result, the load-based schemes would degrade both responsiveness and energy efficiency due to inappropriate or delayed decision on user-interaction intervals.

In order to deal with the inherent limitation of load-based approach, user-driven power management schemes have been proposed [9], [10]. The principle of the user-driven techniques

is to exploit user-level information such as human-computer interaction history for predicting the timing of a user-interaction interval. Though the proposed user-driven techniques could achieve reasonable responsiveness and energy savings, they were not adopted for commercial mobile consumer electronics because the complex prediction mechanism [9] and direct user feedback [10] are impractical for typical mobile systems.

This paper proposes an event-driven processor power management scheme for mobile consumer electronics. In this scheme, an interactive event triggered by a user input (e.g., a screen touch) is directly used to decide the adequate level of the processor performance. A user input can be simply captured as an interactive event by an application framework, since typical mobile systems place framework layer on an OS to serve applications in an event-driven manner. Based on the framework-level support, the proposed scheme achieves high user-responsiveness by providing the highest clock frequency in response to an interactive event, while minimizing ineffective energy consumption by maintaining the lowest clock frequency during non-interactive interval. This event-driven power management assisted by the framework enables mobile consumer electronics to enhance both user experience and energy efficiency without complex prediction and suboptimal heuristic algorithms.

In order to evaluate the effectiveness of the proposed scheme, a latency measurement benchmark, named *LatencyBench*, was developed. LatencyBench measures the time elapsed between a user input triggered and a web browser launch completed. The event-driven power management scheme was implemented as a proof-of-concept prototype in the LatencyBench. With the aid of the LatencyBench and a power meter, user-perceived response time and energy consumption were evaluated for the proposed scheme and various load-based power management techniques on a multi-core mobile CPU. The experimental results showed that the proposed scheme improves user-perceived responsiveness while reducing ineffective energy consumption, compared to the existing schemes that have been adopted in most mobile consumer electronics.

The rest of this paper is organized as follows: Section II introduces existing power management schemes for mobile consumer electronics and the design of the proposed scheme is presented in Section III. Section IV evaluates the prototype implementation of the proposed scheme, and Section V concludes this research.

II. Background and Related Work

This section covers the details regarding the power management schemes in mobile consumer electronics. First, OS-level support structure for processor power management is introduced. Then, the existing processor power management schemes are discussed.

A. OS Support for Processor Power Management

Most modern mobile consumer electronics are equipped with DVFS-enabled mobile CPUs. The implementation details of DVFS such as the maximum and minimum frequencies vary among different processors. In order to cope with the underlying implementation differences, The OS kernels including that of Linux provide an abstract layer for the DVFS control, and then expose a simple interface to the upper layer. Using this interface, a kernel module called *governor* controls the performance level of processors. Governors generally fall into two categories: the static and dynamic governors. The static governors simply maintain a predefined performance level regardless of workloads and system status. On the other hand, the dynamic governors adjust the processor clock frequency on the fly according to the performance demand of workloads.

Along with the DVFS mechanism, processor hotplugging [3] is being widely used for multi-core mobile consumer electronics. In multi-core processors, some cores can be deactivated for reducing energy consumption when the performance demand of a system is low. To measure the performance demand of a system, system load can be used for an OS to decide when to disable over-provisioned cores.

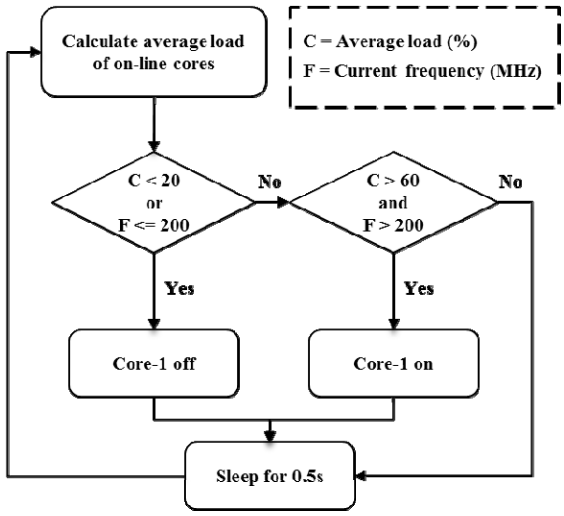

Fig. 1. The load-based processor hotplug algorithm in a commercial dual-core mobile CPU (200 MHz is the lowest supported clock frequency in the target platform)

The load-based processor hotplug algorithm, which is applied to commercial multi-core mobile CPUs, is presented in Fig. 1. This algorithm is usually implemented as a CPU driver to control processor hotplug, and it collaborates with a DVFS governor to control the clock frequency of each core. In order to estimate the appropriate timing for processor hotplug, this algorithm periodically monitors the average load of online cores, and turns off the surplus cores (i.e. Core-1 in a dual core processor) if the average load is less than the low-threshold value. Conversely, Core-1 is turned on when the average load goes beyond the high-threshold value and the current clock frequency is greater than the lowest value. This can be generalized to multiple cores more than two. Considering the critical impact of processor performance on the response time of a workload, a multi-core management policy should be carefully designed as well as DVFS schemes since multi-core

mobile CPUs are widely used in mobile consumer electronics today.

B. Related Work

Most of the existing power management schemes for commercial products are implemented as load-based governors. The *ondemand governor* [11] increases the clock frequency to the highest level when the system load goes beyond a predefined threshold by periodically monitoring the load on the processor. Conversely, the clock frequency is decreased when the processor load becomes below the predefined threshold. In addition, the *conservative governor* [11] was designed for saving more energy than the ondemand governor. To this end, the conservative governor gradually increases the clock frequency when the processor load exceeds the predefined threshold.

The two load-based governors mentioned above make power management decisions reactively based only on processor load changes. In addition, the load monitoring mechanisms used in the two load-based governors would lead to a delayed decision on identifying user-interaction intervals due to its periodic nature. Accordingly, such load-based power management could delay the ramp-up of processor frequency in response to a sporadic user input, thereby degrading user-perceived responsiveness.

To deal with the limitation of the dynamic governors, the *interactive governor* was designed for improving interactive performance. In order to quickly respond to user-interactive load, the interactive governor checks load increase at the time when a processor comes out of idle state; this algorithm assumes that an interactive workload typically makes a transition from idle to active state of a processor. Furthermore, the interactive governor delegates the procedure for increasing the clock frequency to a kernel thread with the highest real-time priority for preventing delayed reaction due to scheduling contention with other tasks. As a result, the interactive governor can quickly increase the clock frequency in response to sporadic user inputs such as screen touches.

The interactive governor, however, cannot immediately ramp-up the clock frequency in response to a user input generated within the non-idle state. This is because the interactive governor still depends on periodic load monitoring in the non-idle state. Moreover, the interactive governor, like the other dynamic governors, can ineffectively increase the clock frequency for a background task, which imposes a high processor load even though the task is irrelevant to user-interactions.

In order to deal with the inherent limitation of load-based approach, some researchers have proposed user-driven schemes of processor power management [9], [10]. The principle of the user-driven techniques is predicting the timing of a user-interaction interval based on explicit user-level information in order to save more energy without compromising high quality of user experiences.

To this end, Zhong *et al.* [9] proposed a technique, which proactively increases the clock frequency before the next predicted user-interaction interval starts using the sophisticated prediction mechanism. Specifically, the proposed prediction mechanism is based on human-computer interaction history and theories from the field of psychology. Though the proposed scheme could reduce ineffective energy consumption without sacrificing user experiences, its complex prediction mechanism prohibits its adoption to commercial mobile consumer electronics.

On the other side, Mallik *et al.* [10] proposed a user-driven frequency scaling (UDFS), which dynamically changes the clock frequency through a direct user feedback mechanism. In order to explicitly identify user-interaction intervals, UDFS requires pressing a button when a user feels uncomfortable with current performance level of a processor. This requirement is impractical for typical users of mobile consumer electronics.

III. Event-Driven Processor Power Management

Unlike desktop PC environments, mobile consumer electronics devices typically allow a single foreground task, with which a user interacts, to run at a time mainly because of the limited display size. In this environment, a user input to a foreground task initiates an interactive event, and the length of time between initiation and completion of the interactive workload induced by the event represents user-perceived response time. Considering the limited battery capacity of mobile consumer electronics, guaranteeing fast responses to the interactive events while minimizing ineffective energy consumption is the ultimate goal of processor power management scheme for high quality of user experiences.

In order to achieve the goal, identifying the initiation and completion of an interactive workload is essential for adaptively adjusting the performance of a processor to the appropriate level. The identification of interactive events can be indirectly inferred by monitoring loads on the processor as in the existing load-based schemes based on the assumption that an interactive event incurs high processor load. However, a CPU-intensive task, which is irrelevant to user-interactions, would lead the load-based scheme to make misjudgment on the initiation and the completion of a user-interaction interval. As a result, the performance level of a processor would be inappropriately adjusted by the load-based scheme.

The interactive event can be directly captured at the application framework layer, which is placed between applications and runtime libraries, to deal with the limitation of the load-based scheme. Basically, the application framework invokes the corresponding functions implemented in an application in response to the events, which are directed to the application so that the application can appropriately handle the events. Hence, the initiation and the completion of a user-interaction interval can be captured by instrumenting framework modules regarding user interface (UI) and display output. For instance, application launch triggered by a user is regarded as the initiation of an interactive workload, and its completion is detected right after the application finishes drawing the necessary components to a display.

The event-driven power management scheme directly exploits the captured interactive events at the application

framework layer in order to adequately adjust the performance level of a processor. Upon receiving an interactive event, the event-driven scheme increases the performance of the processor to the highest level before invoking the corresponding handler function implemented in the application for which the event is destined. After the interactive workload is finished, the performance level of the processor is set to the lowest clock frequency until a next event takes place. By doing so, both the length of time between initiation and completion of the interactive workload (i.e., user response time) and ineffective energy consumption during non-interactive interval can be minimized.

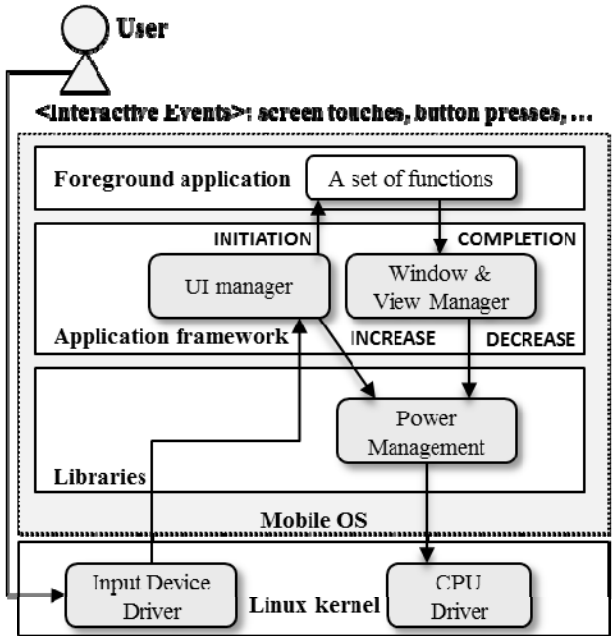

Fig. 2. A schematic view of the event-driven power management scheme

Fig. 2 shows a schematic view of the event-driven power management scheme for mobile consumer electronics. Generally, a user-generated input is firstly delivered to the corresponding input device driver by a hardware unit. Then, the input is eventually forwarded to a UI manager indicating the initiation of an interactive workload. In the event-driven scheme, the UI manager increases the clock frequency of a processor to the highest supported value using a power management library before invoking corresponding functions implemented in the application. Similarly, a window and view manager decreases the clock frequency of a processor to the lowest supported value when the application is ready to handle the next user input after finishing drawing the output associated with the user input.

The proposed event-driven scheme has the following advantages over the existing schemes. Firstly, it imposes little overheads to the system by avoiding periodic monitoring of processor load. Secondly, it eliminates ineffective energy consumption caused by CPU-intensive background tasks irrelevant to interactive events. Finally, it does not depend on a complex prediction mechanism and an impractical user involvement because the proposed scheme directly exploits interactive events triggered by a user to decide the adequate clock frequency.

IV. EVALUATION

This section presents the evaluation methodology and the experimental results. First, the design and implementation of LatencyBench, the response-delay measuring tool, are explained. Second, this section introduces both hardware and software configurations of the experimental platform. Finally, this section analyzes the experimental results based on the introduced evaluation environment. The preliminary analysis results on the effectiveness of existing power management schemes have been presented in the previous work [12].

A. LatencyBench

To assess the power management schemes in terms of user-perceived responsiveness, measurement of response time under each power management scheme is quintessential. To this end, LatencyBench was developed. LatencyBench measures the time interval between the invocation and the completion of the web browser launch activity.

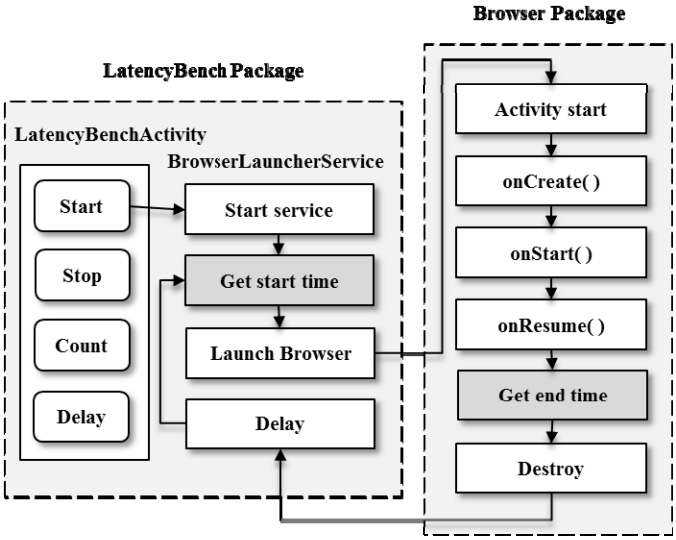

Fig. 3. Execution flow of LatencyBench

LatencyBench was written in Java, and it consists of two packages as shown in Fig. 3. A thread in the LatencyBench package, which was named LatencyBenchActivity, obtains both number of evaluation and time delay between evaluation 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, which records a time stamp when it finished 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.

The proof-of-concept prototype of the event-driven scheme (*event governor)* was implemented with LatencyBench. Unlike the load-based governors, the event governor adjusts the clock frequency to the highest supported value (i.e., 1 GHz) before launching the Browser process. In addition, the event governor adjusts the clock frequency to the lowest supported value (i.e., 200 MHz) right after the Browser process is destroyed by itself and the context is returned.

The identification mechanism for the completion of a user-interaction interval used in the prototype, however, cannot be generally applied to a framework-level implementation. This is because the completion sign of a user-interaction interval is various and depends on the target applications. A framework-level interface, which enables an application to explicitly notify the completion of a user-interaction interval, can be used to deal with this limitation. However, the framework-level support for determination of the user-interaction interval is beyond the scope of this research.

B. Experimental Setup

The prototype implementation was ported to an embedded board, which is being used in lots of commercial smartphones and tablet PCs. The board is equipped with a dual-core mobile CPU operating at up to 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.

The power consumption was measured with an external 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 that is attached to a workstation. The data produced by the power meter are collected by the workstation 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.

The Linux kernel-based mobile OS, which is most widely used for mobile consumer electronics these days, was installed on top of the embedded board. In order to compare the event-driven scheme with various power management schemes, several Linux power management governors were incorporated into the experimental platform. In addition to the dynamic governors (i.e., the load-based governors) addressed in Section II, two static governors were used for the evaluation; the *performance governor* keeps the highest clock frequency whereas the *powersave governor* statically sets the frequency to the lowest one.

The experimental system was also configured with the three multi-core management policies. The first policy is *DYNAMIC,* which uses the load-based hotplug algorithm introduced in Section II. As the second policy, the system was set to use only one of the two cores. This is dubbed as *SINGLE*. In the third policy, *DUAL*, both cores are always activated.

C. Experimental Results

To measure the performance in terms of response time, ten consecutive launches of the web browser were carried out using LatencyBench, and then the recorded response times were averaged for comparison. Note that the total execution time of the benchmark was different depending on the applied power management scheme. Hence, the energy consumption under each scheme during the idle interval with display-on was padded according to the difference of the total execution time to the slowest one.

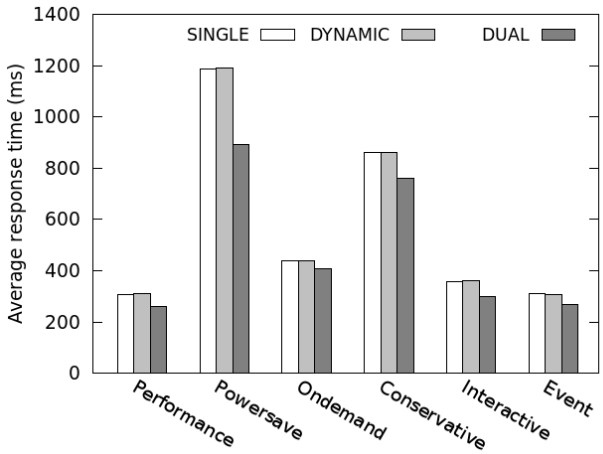

Fig. 4. Average response time of various power management schemes using LatencyBench

Fig. 4 shows the average response time of various combinations of DVFS and multi-core management schemes using LatencyBench. The event governor achieved better responsiveness than any other load-based governors because the load-based governors reactively increased the clock frequency after processor load increased. Moreover, the event governor in combination with DUAL showed only about 3% degradation in responsiveness as compared to the performance governor. Note that the response time under DYNAMIC was close to that under SINGLE though both cores are supposed to be in activation states as processor load increases under DYNAMIC.

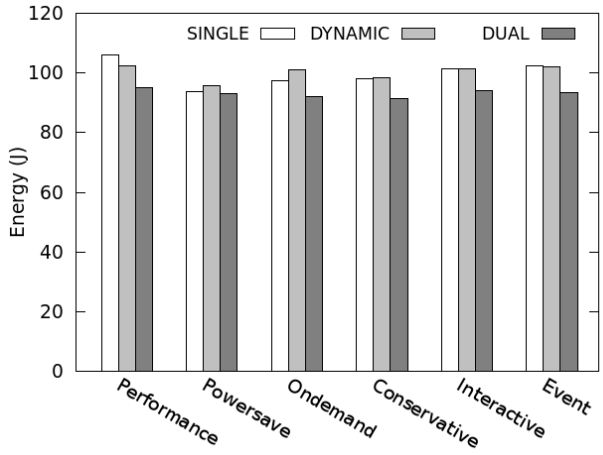

Fig. 5. Energy consumption of various power management schemes during the execution of LatencyBench

Fig. 5 shows the energy consumption during the execution of LatencyBench when the various power management schemes were applied. The proposed event governor showed

similar energy consumption to that of the load-based governors. In addition, the event governor achieved lower energy consumption than the performance governor since the event governor maintained the lowest clock frequency during non-interactive intervals. Note that all the governors in combination with DUAL consumed less energy than those with SINGLE and DYNAMIC. This is because DUAL produced the longest idle intervals among the three core management policies due to the fastest execution.

In typical mobile systems, background tasks are concurrently executed with a foreground task, and some of the background tasks incur high processor load. Hence, the load-based schemes would fail to identify user-interaction intervals in such a case, since the load-based schemes only use processor load as a metric for identifying user-interaction interval. In order to validate this scenario clearly, additional experiments were conducted. In the following experiments, a CPU-intensive background task was configured to run concurrently with LatencyBench.

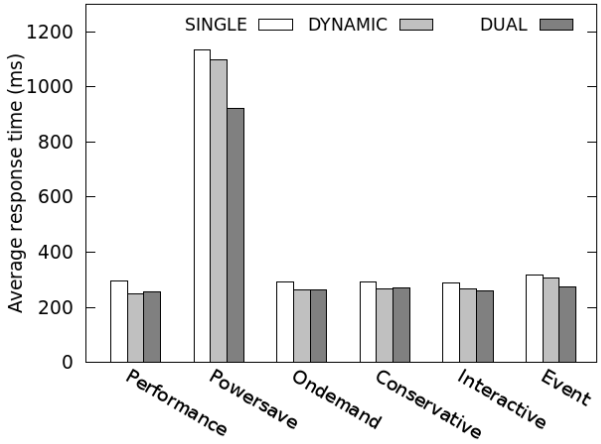

Fig. 6. Average response time of various power management schemes using LatencyBench with a CPU-intensive background task.

Fig. 6 presents average response time of the experiments with the CPU-intensive background task. The load-based governors showed nearly identical response time to that of the performance governor since the load-based governors maintained the clock frequency to the highest level throughout the evaluation due to the high processor load from the background task. The performance of the event governor was similar to that of the performance governor. As opposed to the previous experiments, which were conducted without the background task, DYNAMIC in combination with the performance and load-based governors, achieved the same performance level to that of DUAL. This is because the second core was always activated by the background task, which pushed processor load beyond the threshold to turn on the second core under DYNAMIC. On the other hand, the second core was activated only during the Browser task was being executed under the event governor because it maintained the lowest clock frequency in the idle intervals.

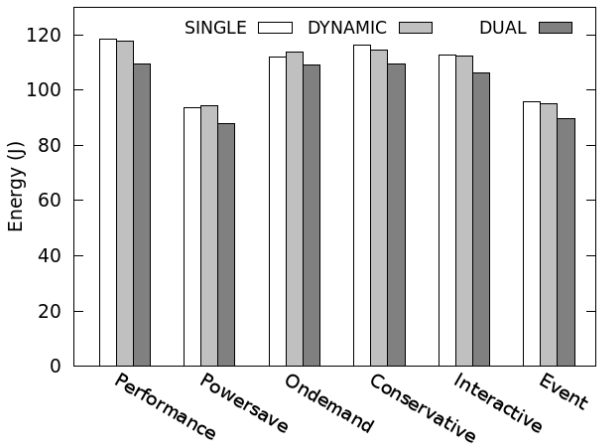

Fig. 7. Energy consumption of various power management schemes during the execution of LatencyBench with a CPU-intensive background task.

Fig. 7 shows the energy consumption during the execution of LatencyBench with the CPU-intensive background task. The performance and load-based governors consumed the similar amount of energy since the load-based governors showed the identical behavior to the performance governor due to the constantly high processor load from the background task. On the other hand, the event governor consumed the similar amount of energy in comparison to that of the powersave governor. This is because the event governor uses interactive events, instead of processor load, as an indicator for performance demand. By maintaining the highest clock frequency only during the user-interaction intervals, the event governor achieved the highest energy efficiency among the evaluated governors.

Overall, the proposed event governor consumed about 20 % less energy than the performance governor in the DUAL case with only about 5 % increase in average response time. The difference in response time between the event and performance governor was about 17 ms, which is negligible amount since a user cannot perceive the time difference [13]. Note that the event governor could not benefit from DYNAMIC in terms of both responsiveness and energy efficiency. This result is consistent with the experimental results without the background task. The overall results on DYNAMIC imply that it cannot effectively utilize the computation power of the multi-core processor.

D. Analysis on Dynamic Processor Hotplug

Similar to the server and desktop PC environments, multi-core processors are commonly adopted to mobile consumer electronics today. It is believed that the multi-core architecture is an energy-efficient way to improve performance in comparison to raising clock frequency of a single-core processor. However, the increased power consumption from integrating multiple cores may harm the energy efficiency rather than improve depending on the system and workload status. Therefore, an efficient multi-core management scheme is crucial in order to improve energy efficiency without

sacrificing user experience. According to the experimental results, however, the load-based scheme of processor hotplug failed to fully utilize the performance benefit of the dual-core processor in response to a sporadic user input. In order to further analyze the problem on the load-based processor hotplug, scheduling delay during the experiments was measured.

TABLE I
AVERAGE WAIT TIME OF THE BROWSER PROCESS IN THE RUN-QUEUE UNDER DIFFERENT MULTI-CORE MANAGEMENT POLICIES WITH THE PERFORMANCE GOVERNOR

Multi-core Management Policy	Average Wait Time (us)
SINGLE	907
DUAL	282
DYNAMIC	903

Table I shows the average wait time of the Browser process in the run-queue when the three management policies were applied in combination with the performance governor. The average wait time of the DUAL case was significantly shorter than that under both SINGLE and DYNAMIC. In addition, the average wait time of DYNAMIC was very similar to SINGLE, which again means that DYNAMIC could not take advantage of the second core. These results suggest that the DYNAMIC algorithm, with its default parameters, fails to rapidly react to a sporadic user input.

To further explore the possible improvement of hotplug effectiveness, additional experiments were conducted by varying values of three configurable parameters of the DYNAMIC 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 interval that determines the granularity of the decision cycle.

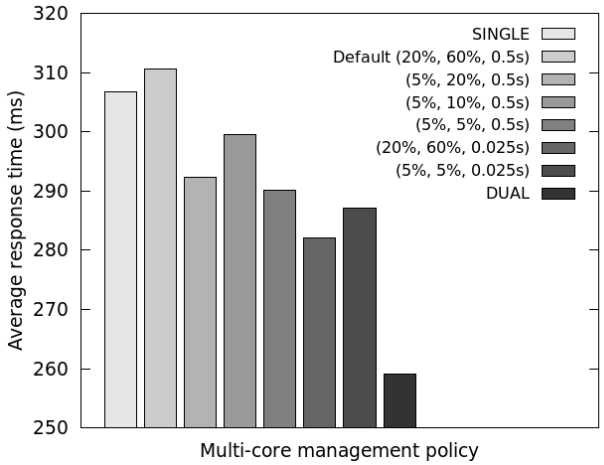

Fig. 8. Average response time under various multi-core management policies in combination with the performance governor; (x %, y %, z s) = (low threshold to turn off, high threshold to turn on, monitoring interval).

Fig. 8 shows the average response time according to various configurations of DYNAMIC. The average response time of DYNAMIC 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 processor load. Moreover, the average response time was further reduced than the other configurations when the monitoring interval was shortened to 0.025 s. This result implies that the monitoring interval is more critical to the user-perceived response time than the threshold values in the load-based algorithm of processor hotplug.

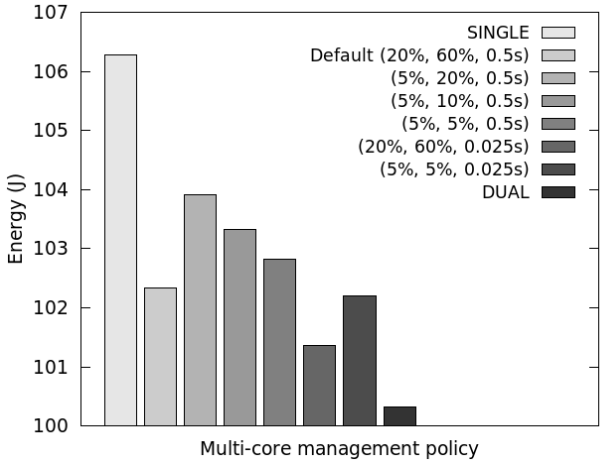

Fig. 9. Energy consumption under various multi-core management policies in combination with the performance governor during the execution of LatencyBench; (x %, y %, z s) = (low threshold to turn off, high threshold to turn on, monitoring interval).

Fig. 9 shows energy consumption under various configurations of the multi-core management policies with the performance governor. The energy consumption under the DUAL setup was lower than that under the other policies because DUAL finished the workload in the shortest time and stayed in the longest idle state. The energy consumption under DYNAMIC was reduced as the parameters were changed to the aggressive values. Overall, the difference of response time between the best and the worst case was 20 % whereas the difference of energy consumption was only 6 %.

The analysis results on the effectiveness of the multi-core management policies revealed that DYNAMIC has little contribution to energy savings. Moreover, DYNAMIC could not utilize the additional processor core to improve user-perceived responsiveness. Though several customizations on the parameters could help to mitigate the problem, DYNAMIC with the various parameters could not perform as fast as DUAL. This limitation arises from the inherent delay of processor hotplug which is comprised of both hardware and software latency [14]; the hotplug latency was about 100 ms in the evaluated platform. This inherent scheduling delay cannot be eliminated by any customizations of the parameter values. Therefore, a multi-core management scheme should consider the inherent hotplug latency in order to save more energy without hurting user-perceived responsiveness.

V. CONCLUSION AND FUTURE WORK

This paper proposed the event-driven power management scheme, which exploits interactive events triggered by user

inputs for adjusting processor performance. The proof-of-concept prototype was implemented for validating the effectiveness of the event-driven power management scheme. The evaluation results showed that the proposed scheme achieved similar responsiveness to the best case of the conventional load-based power management schemes while saved approximately up to more than 20 % of energy when mixed with a CPU-intensive background task. The experimental results also showed that the use of a simple processor hotplug algorithm might harm user-perceived responsiveness because it cannot timely react to sporadic user inputs.

Although the evaluated prototype successfully showed the effectiveness of the event-driven approach, the current scheme cannot be blindly applied to the existing applications since the completion sign of a user-triggered event cycle could be extremely diverse depending on the target applications, and thus it is difficult to accurately determine the performance-demanding time interval. In order to deal with this limitation, the authors are working on the framework-level support that helps the event-driven power manager to determine the duty cycles from user-triggered events.

REFERENCES

[1] Trevor Mudge, "Power: A first class architectural design constraint," *IEEE Computer,* vol. 34, no. 4, pp. 52-58, Apr. 2001.

[2] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *Proc. 1st USENIX Conf. Operating Systems Design and Implementation,* Nov. 1994.

[3] Z. Mwaikambo, A. Raj, R. Russel, and J. Schopp, "Linux kernel hotplug CPU support," in *Proc. Ottawa Linux Symp.,* Jul. 2004.

[4] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. 18th ACM Symp. Operating Systems Principles,* Oct. 2001.

[5] J. R. Lorch and A. J. Smith, "Operating system modifications for task-based speed and voltage scheduling," in *Proc. 1st Int. Conf. Mobile Systems, Applications, and Services,* May 2003.

[6] E. Seo, S. Park, J. Kim, and J. Lee, "TSB: A DVS algorithm with quick response for general purpose operating systems," *J. Syst. Architecture,* vol. 53, no. 1-2, pp. 1-14, Jan.-Feb. 2008.

[7] S. Lim, S. W. Lee, B. Lee, and S. Lee, "Power-aware optimal checkpoint intervals for mobile consumer devices," *IEEE Trans. Consumer Electron.*, vol. 57, no. 4, pp. 1637-1645, Nov. 2011.

[8] M. Kim, J. Kong, and S. W. Chung, "Enhancing online power estimation accuracy for smartphones," *IEEE Trans. Consumer Electron.*, vol. 58, no. 2, pp. 333-339, May 2012.

[9] L. Zhong and N. K. Jha, "Dynamic power optimization targeting user delays in interactive systems," *IEEE Trans. Mobile Comput.,* vol. 5, no.11, pp. 1473-1488, Nov. 2006.

[10] A. Mallik, B. Lin, G. Memik, P. Dinda, and R. P. Dick, "User-driven frequency scaling," *Comput. Architecture Lett.,* vol. 5, no. 2, pp. 1-4, Dec. 2006.

[11] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proc. Ottawa Linux Symp.,* Jul. 2006.

[12] S. Kim, H. Kim, J. Kim, J. Lee, and E. Seo, "Empirical analysis of power management schemes for multicore smartphones," accepted for publication in *Proc. ACM Int. Conf. Ubiquitous Inform. Manage. and Commun.*, Jan. 2013.

[13] N. Tolia, "Quantifying interactive user experience on thin clients," *IEEE Computer,* vol. 39, no. 3, pp. 46-52, Mar. 2006.

[14] S. Panneerselvam and M. M. Swift, "Chameleon: Operating system support for dynamic processors," in *Proc. 17th Int. Conf. Architecture Support for Programming Languages and Operating Systems,* Mar. 2012.

BIOGRAPHIES

Sangwook Kim received his B.S. degree in Computer Engineering from Sungkyunkwan University (SKKU) in 2010 and M.S. degree in Mobile Systems Engineering from SKKU in 2012. He is currently a Ph.D. candidate in the IT Convergence Department, SKKU. His current research interests include operating systems, virtualization, cloud computing, and embedded systems.

Hwanju Kim received his B.S. degree in Information and Computer Engineering from Ajou University in 2006 and M.S. degree in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 2008. He is currently a Ph.D. candidate in the Computer Science Department, KAIST. His research interests include virtual machine, embedded systems, and storage systems.

Jeaho Hwang received his BS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 2007, and MS degree in computer science from KAIST in 2009. He is currently a PhD candidate in the Computer Science Department, KAIST. His current research interests include operating systems, system reliability, virtualization, and embedded systems.

Joonwon Lee received his B.S. degree in Computer Science from Seoul National University in 1983 and M.S. and Ph.D. degrees from the Georgia Institute of Technology in 1990 and 1991, respectively. He is currently a professor in Sungkyunkwan University (SKKU). Before joining SKKU, he was a professor at the Korea Advanced Institute of Science and Technology from 1992 to 2008. His current research interests include low power embedded systems, system software, and virtual machines.

Euiseong Seo received his BS, MS, and PhD degree in computer science from KAIST in 2000, 2002, and 2007, respectively. He is currently an assistant professor in college of ICE at Sungkyunkwan University, Korea. Before joining Sunkyunkwan University in 2012, he had been an assistant professor at UNIST, Korea from 2009 to 2012, and a research associate at the Pennsylvania State University from 2007 to 2009. His research interests are in power-aware computing, real-time systems, embedded systems, and virtualization.