2016 10th International Conference on Next Generation Mobile Applications, Security and Technologies

A Lightweight Pseudo CPU Hotplug Mechanism for Mobile Devices

Kyoung Don Jang[†§], Dong Hyun Kang[†], Do Hyoung Kim[§], Hyun Jin Park[§], and Young Ik Eom[†]
[†]Sungkyunkwan University
Suwon, Korea
Email: {alex.jang, kkangsu, yieom}@skku.edu
[§]Samsung Electronics
Suwon, Korea
Email: {dh0703.kim, grant.park}@samsung.com

***Abstract*—Power efficiency has become one of the most important issues in mobile devices because of their limited battery size. However, many mobile applications and processors negatively accelerate power consumption of mobile devices. Previous work (e.g., CPU hotplug) may be inappropriate in mobile devices due to their slow responsiveness because it has to migrate all processes running on a CPU, which is to be turned off to save power consumption, to another CPU, thereby resulting in slow responsiveness. In this paper, we propose a novel mechanism, called *pseudo CPU hotplug*, which efficiently guarantees fast responsiveness while reducing the power consumption of mobile devices. Our mechanism eliminates almost all overhead of operation system such as that of synchronous task migration, and leaves only architecture-dependent overhead of entering deep idle mode. Our evaluation results clearly show that *pseudo CPU hotplug* outperforms the traditional CPU hotplug mechanism by up to 997 times in terms of response latency. In addition, our mechanism increases the time spent in deep idle mode more than 30% compared to normal status.**

I. Introduction

In mobile devices (e.g., MP3 player, laptops, tablet PCs, and smart phones), power efficiency is one of the most important issues due to limited battery size. Unfortunately, most mobile applications negatively affect energy efficiency of the mobile devices by employing a lot of hardware services, including GPS, WiFi, and Bluetooth. In addition, emerging mobile processors exacerbates power consumption of the devices because they require high clock frequency (e.g., 2.3 GHz).

In the past years, many researchers and developers have focused on reducing the power consumption in data center and desktop environments. Also, several studies proposed CPU technologies that reduce power consumption caused by mobile processors. DVFS(Dynamic Voltage Frequency Scaling) and CPU hotplug technology are widely used to control the power consumption of mobile systems [1], [2]. To efficiently reduce the power consumption, DVFS technology adjusts both the frequency and voltage of processors at runtime by periodically monitoring utilization of processors. On the other hand, CPU hotplug technology improves the power efficiency by disabling CPU cores without system reboot. Unfortunately, CPU hotplug technology may be inappropriate in mobile environments because they have negative effect on the mobile devices in terms of responsiveness. Especially, CPU hotplug can consume a large amount of energy in a short time because it must migrate all processes running on a CPU, which is to be turned off for power optimization, to another CPU. Therefore, the responsiveness of CPU hotplug can be dropped according to the number of threads running on the CPU [3]. To understand the impact on the responsiveness of mobile devices when using the CPU hotplug, we first measured CPU hotplug latency by varying the number of threads from 1 to 50. Figure 1 shows the latency of the CPU hotplug technology on real smart phone, Samsung Galaxy Note5. As shown in Figure 1, the latency increases as the number of threads increases.

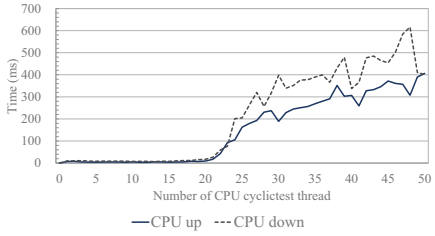


Fig. 1. CPU hotplug latency with cyclictest threads on Galaxy Note 5

Some researchers also proposed OS independent technologies to manage the power consumption of mobile devices. ACPI(Advanced Configuration and Power Interface) is an open industry specification for power management. ACPI is designed as a stackable deep idle mode, called Cn state. For ACPI, Linux supports C1-Cn states, which represent idle CPU power states. Devices can save more processor power as Cn state increases [4] (i.e., the power consumption of the deepest Cn state is possibly similar to that of power-off state).

In order to efficiently manage power consumption of mobile devices, in this paper, we propose a lightweight *pseudo CPU hotplug* mechanism that is designed to take the following goals:

- *Transparency* to provide similar interface to the conventional CPU hotplug technology
- *Low power consumption* to reduce power consumption of the mobile devices
- *Fast responsiveness* to guarantee quick system response for mobile applications after the deep idle mode

2161-2897/16 $31.00 © 2016 IEEE
DOI 10.1109/NGMAST.2016.14

TABLE I
C-STATE IN LINUX KERNEL FOR ARM CORE

state	ARM idle mode	ACPI Spec.	CPU logic	RAM array	Wake-up Mechanism
C0	RUN MODE	Operating state (executing instructions)	Powered up, Everything clocked	Powered	N/A
C1	STANDBY	Halt (not executing instructions)	Powered up, Only wake-up logic clocked	Powered	Wake up on interrupt (external I/O, Timer/Watchdog), L1 memory system only wakes up temporally to process SCU coherency request
C2	DORMANT	Stop-clock (clock is stopped)	Powered off	Retention state voltage only	Via external wake-up event from power controller
C3	SHUTDOWN	Sleep (cache coherency disabled)	Powered off	Power off	Via external wake-up event from power controller

The remainder of this paper is organized as follows. We explain the background to understand how Linux handles the ACPI specification in Section 2. Section 3 describes our motivations and Section 4 shows *pseudo CPU hotplug* mechanism in detail. Then, we present our evaluation results in Section 5 and we conclude the paper in Section 6.

II. BACKGROUND

A processor state can simply be classified into two modes: *online* when the CPU core operates and is being used to process tasks, and *offline* when the CPU core is entirely powered down and thus cannot be used to process tasks. Conventional CPU hotplug mechanism switches the CPU state between online and offline. And DVFS categorizes online mode to several states to represent the current CPU core's characteristic. This is because the state power consumption of processor varies depending on the CPU utilization, frequency, and voltage. This section covers the processor state for power management.

A. Processor power management

ACPI is an open industry specification for power management on electronic systems. ACPI defines not only power states of entire electronic system, but also power states of processor. Especially, ACPI focuses on processor power management, because it is the key ingredient in system power management. Each of processor power states has different frequencies and voltages that affect the power consumption. Figure 2 shows ACPI-defined processor power states [5]:

1) G0/S0: System working state. System component including processor is running.
2) G1 ˜ G3: System wide sleeping states. In these states, operation system is inoperative. Specially in G3 state, only RTC battery is the power consumption component.
3) Processor P-state(P0 ˜ Pn): Performance states. These states represent different processor frequencies and voltages. DVFS technology is reliant on P-state. Deeper P-state represents low frequency, low voltage and less power consumption(power consumption depends on the square of voltage).
4) Processor T-state: Throttle state. Only processor frequency is changed, whereas voltage is unchanged in this state. Linux kernel thermal framework uses T-states for (emergency) throttling [2].
5) Processor C-state:(C0 ˜ Cn): C-states are categorized into two stages. The first stage is a C0 state, which is a processor running state. This state is a superset of P-state and T-state. The second stage includes from C1 to Cn state. This stage represents the processor states of idle CPU core. Each C-state shows the power status of processor components such as CPU logic, RAM array, SCU and so on. This specification for C-state can be implemented various ways. In Linux kernel for ARM core, C-state is implemented from C0 to C3 by architecture-dependent idle modes, which are called 'standby mode' and 'dormant mode'. Table I illustrates the implementation of C-state in Linux kernel for ARM core [6].

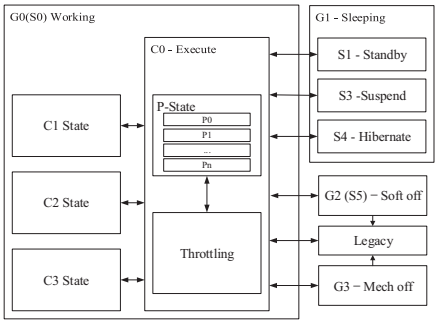


Fig. 2. An ACPI power state diagram

III. MOTIVATION

This section describes the conventional CPU hotplug mechanism and the current usage of processor idle power state - called the deep idle modes. We analyze the conventional CPU hotplug mechanism to find some points to improve. Also we measure the current deep idle modes duration in idle state to figure out its usage.

A. Conventional CPU hotplug

Conventional CPU hotplug scheme consists of two steps: *hotplug out* where a CPU moves from online to offline and *hotplug in* where a CPU moves from offline to online.

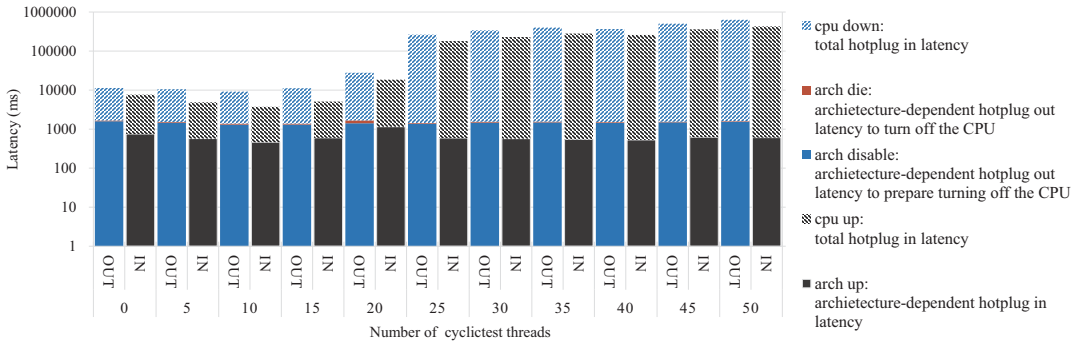


Fig. 3. CPU hotplug steps with cyclictest threads

Hotplug out: Linux kernel executes `cpu_down()` function for hotplug out. It can be divided into two parts. One is the architecture-independent part(OS handling part), where it deals with task migration, IRQ handling, kernel thread management, and so on.

1) Task migration: Migrate all running tasks on the target CPU to another online CPU.
2) IRQ handling: Disable IRQs that are dedicated target CPU(e.g., local timer interrupt) and migrate general IRQs toward the target CPU.
3) Kernel thread management: Manage kernel threads which are running on the target CPU by creating/deleting or parking/unparking [2]. Creating and deleting of kernel threads may add a few seconds to the CPU hotplug in/out [7]. Therefore, latest Linux kernel has adopted parking kernel thread instead of deleting it. In other words, kernel threads remain runnable and are bound to an offline CPU; they are called idle threads.
4) Extra work: Manage CPU related variables in memory such as `cpu_online_mask`, stop watchdog daemon which is for soft or hard lockup detection, clear timer event for target CPU such as *hrtimer* for watchdog daemon registration, and carry out miscellaneous jobs.

The other is architecture-dependent part that makes actual CPU go offline by executing shutdown commands for each architecture such as `platform_cpu_die` in Linux kernel. In this part, Linux kernel performs the followings:

1) Check the CPU whether it can be offline or not
2) Complete the current shutdown request
3) Prepare to enter the power down state(i.e., cache flush)
4) Take a loop in the power down state until the hotplug in commands are invoked
5) Leave the power down state by wake up command

Hotplug in: Linux kernel executes `cpu_up()` function in hotplug in. It can also be divided into the architecture-independent part and the dependent part. Architecture-independent part also deals with IRQ handling and kernel thread management. But task migration is not invoked during hotplug in. Actually, it is done at schedule time. Kernel scheduler assigns tasks to the target CPU according to the load balancing policy.

1) IRQ handling: Enable IRQs that are dedicated to target CPU; Unmask interrupt bit for interrupts controller.
2) Kernel thread management: Create kernel threads if they have been deleted during hotplug out; Unpark kernel threads if parked kernel threads are runnable
3) Extra work: Manage CPU related variable in memory and wake watchdog daemon up. Carry out miscellaneous jobs.

We measured each step of CPU hotplug in and out with Linaro's tracing patches [4]. Table II and Figure 3 show that conventional CPU hotplug latency. In this table and graph, we get the hotplug latency for architecture-dependent part is not affect by the number of threads and is smaller than the latency for architecture-independent part.

TABLE II
HOTPLUG LATENCIES FOR EACH STEP

step	min	max
cpu up	3280us	406023us
arch up	445us	3143us
cpu down	6326us	615877us
arch disable	1194us	1593us
arch die	48us	821us

B. Current usage of the deep idle modes

To better understand the deep idle modes (i.e., C1 and C2 states), we first measured how long do processors stay in the deep idle modes. For fair comparison, we calculated the time spent in the deep idle mode of each core except CPU 0 because interrupt handling of CPU 0 can pollute our evaluation results. Unsurprisingly, we found that traditional hotplug mechanism aggressively employs the deep idle mode to save the power consumption of mobile devices. All cores except CPU 0 spend 90.28% of their time in the deep idle mode. In addition, we observed that the deep idle mode is triggered for a short period of time. Table III illustrates more details about entrance attempt, the time spent in the deep idle mode, and ratio on Galaxy Note 5 without user input.

Above observations show that the conventional CPU hotplug mechanism takes most of the time in the architecture-independent part and the deep idle mode can be entered and exited with a small latency. These situations motivate us to design a new mechanism.

TABLE III
DEEP IDLE MODE ON GALAXY NOTE 5 WITHOUT USER INPUT

Total experiment time		318927179us	100%
State	Entrance attempt	Average time spent in the deep idle mode	Percentage
The time spent in the deep idle mode excluding CPU 0	13180	287951702.8us	90.28%
The time spent in the deep idle mode including CPU 0	15108	279866887.3us	87.75%

IV. DESIGN

In this paper, we propose a new approach, called the *pseudo CPU hotplug*, which reduces the CPU hotplug latency by employing the deep idle mode. In addition, our approach can minimize the overhead of kernel threads management by employing the *parking* algorithm that maintains the kernel threads on the main memory instead of deleting [2].

The design goals of our mechanism are as follows:

1) Stability and robustness for the deep idle state
2) Low state latency
3) Transparency

To achieve all the goals, we propose an asynchronous approach for task migrations (Section IV-B), a simplified notification process (Section IV-E), a modified kernel thread management (Section IV-D), and a screening approach to maintain the deep idle mode (Section IV-C). In addition, we offer a generic interface similar to the current CPU hotplug mechanism in order for our mechanism to be used easily (Section IV-A).

A. Generic interface for user applications

Most of the time, conventional CPU hotplug mechanism is a passive operation, where it is triggered by user input. Linux kernel provides sysfs node for this purpose. Our proposed mechanism provides a similar interface to user application (e.g., `/sys/devices/system/cpu/cpuX/pseudo_online`). Once a user application writes to the sysfs node for *pseudo hotplug out*, the target CPU core enters to the deep idle mode automatically. This interface provides an abstract layer for user applications. They do not need to concern about internal operations for hotplug.

B. Asynchronous task migration

Conventional CPU hotplug mechanism simultaneously migrates all applications during the task migration process. For task migration, applications running on a CPU should be stopped and moved to another online CPU. Therefore, if there are a lot of running processes on the CPU, the conventional CPU hotplug has to take a long time for the task migration. To avoid this bottleneck, we designed an asynchronous task migration mechanism. Since *pseudo hotplug* simply sets the *pseudo online mask* of each CPU to 1 instead of turning the CPU off when CPU hotplug is triggered, tasks can be maintained on the CPU. Therefore, asynchronous task migration

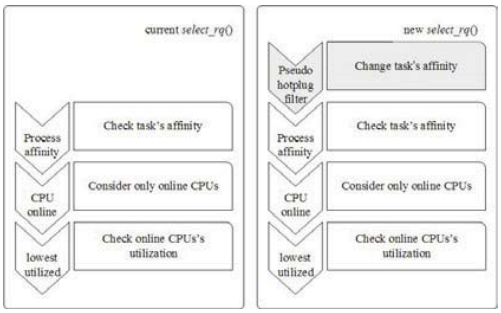


Fig. 4. A diagram to show how to select a CPU to run (select_rq)

migrates each task, which remains on the CPU, when each task is rescheduled (i.e., `select_rq()` in Linux kernel.). Figure 4 shows this routine briefly. Scheduler selects the destination CPU in consideration of the 3 conditions when it picked the task to run, where they are the picked task's affinity, the online CPU status, and the CPU utilization. Our proposed mechanism adds one more condition, *pseudo online mask*. Scheduler checks the mask before the task's affinity and prevents to select the pseudo offline CPU. The tasks will be migrated to the pseudo online CPU.

C. Extension of the deep idle modes

We tried to extend the time spent in the deepest idle mode on a pseudo offline CPU. First, our mechanism removes the checking routine to enter the deepest idle mode. When *pseudo hotplug* is triggered, we force the target CPU to enter to the deep idle mode instead of turning off the CPU. Normally, CPU core enters the deep idle mode from lowest state C1 and then goes to deeper states. To reduce latency entering the deeper idle mode, we make the fast check routine to determine which idle mode to enter. In pseudo hotplug mechanism, all CPU cores directly enters the deepest idle mode without entering other idle modes when the CPUs are in the pseudo offline state. Therefore, our mechanism avoids wasting time to change the idle modes.

Second, our mechanism tries to block the interrupts as much as possible. Since there are many wake up interrupts such as migration functions to another processor, communication between processors(IPI) in Linux kernel, we modified the interrupt handler to receive only critical IPI signals like `IPI_CPU_STOP`. We also modified some routines for kernel threads like watchdogd because it periodically wakes up the CPU to register the local timer. As a result, we can maintain CPU idle states as long as possible.

D. Kernel threads management

Parking and unparking kernel threads are significant improvement in conventional hotplug mechanism [2]. Our mechanism also adopts this algorithm when we switch between pseudo online and offline states. We can minimize the running states of the kernel threads with the parking algorithm.

E. Simple notification

Our proposed mechanism notifies when the CPU states are changed. These notifications are simpler than those of the conventional hotplug mechanism. Conventional hotplug mechanism has many notifications such as `CPU_UP_PREPARE`, `CPU_STARTING`, and `CPU_ONLINE` for the hotplug in. This is because CPU hotplug can be triggered on any CPU core, but some of the codes should be executed on the target CPU. For example, `CPU_STARTING` should be handled on the target CPU in Linux. On the other hands, pseudo CPU hotplug mechanism needs a few notifications: `PSEUDO_CPU_ONLINE` and `PSEUDO_CPU_OFFLINE`. These notifications can be executed on any CPU because our mechanism force the target CPU to enter the deep idle mode at once and it does not turn the target CPU off.

V. Evaluation

Our evaluation targets the following questions:

- How long does it take for hotplug to complete?
- Does latency depend on number of process?
- Is pseudo CPU hotplug latency good for dynamic use?
- Does pseudo CPU hotplug mechanism take more power consumption than current hotplug mechanism?

To get answers to these questions, we performed three kinds of experiments. The first is the measurement for latency (Section V-A), the second is the measurement for power consumption (Section V-B), and the last is the measurement for the time spent in the deep idle mode (Section V-C). All experimental results were collected from Samsung Galaxy Note 5. Table IV illustrates more details about Samsung Galaxy Note 5.

TABLE IV
Samsung Galaxy Note 5 Specification

Kernel version	3.10.61
Android Version	5.1.1
Processor	exynos7420 SoC, ARM CortexA57 MP4 2.1GHz CPU + ARM Cortex-A53 MP4 1.5GHz CPU
Memory	4GB LPDDR4 SDRAM
Display	5.7 inch WQHD AMOLED

A. Measurement of the latency

We evaluated the hotplug latencies with the conventional mechanism and the proposed mechanism. First of all, we defined the hotplug latencies as follows:

- The latency of conventional CPU hotplug out: It starts from user input such as writing sysfs node and ends right before the CPU power down command is invoked.
- The latency of conventional CPU hotplug in: It starts from user input such as writing sysfs node and ends when the `CPU_ONLINE` notification is completed.
- The latency of pseudo CPU hotplug out: It starts from user input such as writing sysfs node and ends right before executing command to enter the deep idle mode.
- The latency of pseudo CPU hotplug in: It starts from user input such as writing sysfs node and ends when the `CPU_PSEUDO_ONLINE` notification is completed.

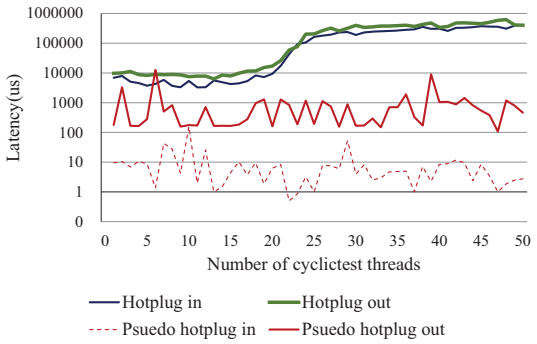


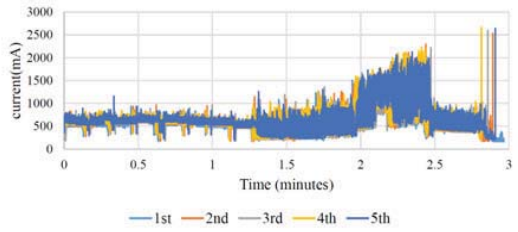
Fig. 5. hotplug latencies with cyclictest threads

We modified the Linux kernel to add logs to measure each start and end time. Linaro has similar patches for logging hotplug latencies. We applied these patches for conventional hotplug mechanism and added more tracing points for our proposed mechanism. To reduce environmental variation, we waited until Android home screen showed up. Then we turn on the airplane mode to minimize the variation by modem. Modem's power consumption varies depending on signal strength, but we wanted to focus on CPU power consumption at this experiment [8]. Also, we fixed the brightness of the display for the same purpose. Finally, we executed conventional CPU hotplug mechanism and our proposed mechanism. We ran each experiment with increasing number of cyclictest threads to stimulate CPU load situation [9]. We made from 1 cyclictest thread to 50 cyclictest threads on each experiment, measuring each hotplug latency.

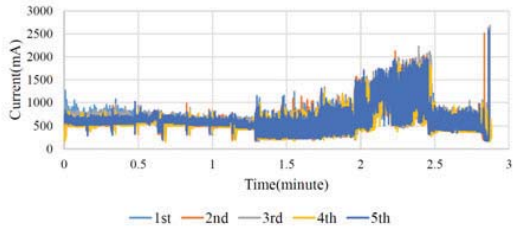
Figure 5 shows that the *pseudo CPU hotplug* mechanism achieves notable reduced latency compared to the conventional CPU hotplug mechanism. The *pseudo CPU hotplug* mechanism outperforms the conventional CPU hotplug mechanism by up to 997 times in terms of response latency. The latencies of the conventional CPU hotplug mechanism increase according to the number of cyclictest threads. But the latencies of the proposed mechanism are independent from the number of running processes, as we expected. The conventional CPU hotplug mechanism handles all the running processes on the target CPU. These processes should be migrated or parked at once. But the proposed mechanism migrates the tasks asynchronously, and it reduces some delays induced by notifications and checking steps to enter the deep idle mode. Briefly, we can remove most of the OS-level overhead during hotplug in and out.

B. Measurement of the power consumption

We measured the power consumption of the conventional hotplug mechanism and the proposed mechanism. We used the AnTuTu, which is a well-known Android benchmark program. This benchmark program consists of the CPU, memory, I/O, and GPU test scenarios and has the characteristics of high utilization of each component to make it possible to measure the maximum performance. To observe the runtime power consumption, we used the power monitor from the Monsoon



(a) Conventional CPU hotplug



(b) Pseudo CPU hotplug

Fig. 6. A current measurement with running AnTuTu benchmark

Solutions Inc.. Power monitor contains the DC power supply to provide a voltage and current to the test device. It gathers sampling current data(mW) with a voltage information lively. We connected the Galaxy Note 5, the Samsung anyway jig box, power monitor, the desktop PC in a row. After connecting all these devices, we set the input voltage to 4V, which is normally provided by internal battery. In this way, we could observe the power consumption of the entire system.

Figure 6 and Table V show the power consumption (current) of the conventional CPU hotplug mechanism and the proposed mechanism.

TABLE V
A POWER CONSUMPTION WITH RUNNING ANTUTU BENCHMARK

	Power Consumption(mW)		
Method	Min	Max	Avg
Conventional hotplug mechanism	2426.130889	2664.934272	2565.944851
Proposed mechanism	2439.704118	2677.566341	2571.707039

This experiment shows that the conventional hotplug mechanism and the proposed mechanism have less than 1% difference at the entire power consumption for the total execution time. As we expected, the total power consumption of the proposed mechanism is larger than that of the conventional hotplug mechanism. But this experiment shows that each power consumption of offline processor and for deep idle mode does not significantly affect the entire system power under the high CPU utilization.

C. Measurement of the time spent in the deep idle mode

Finally, we measured the time spent in deep idle mode during pseudo offline and normal status. We used the logging tool called *CPU profiler* which is already integrated in the Galaxy Note 5's kernel source by S.LSI SoC team. The *CPU profiler* offers the time spent in the deep idle mode. We compared normal status and pseudo offline status with the touch gesture in Android home screen. Table VI illustrates the time spent in the deep idle mode and the ratio.

TABLE VI
THE TIME SPENT IN THE DEEP IDLE MODE AND RATIO WITH THE SCREEN SCROLLING

Status	Total run time (us)	The time spent in the deep idle mode (us)	ratio
Normal	4848095562	3054300274	63%
Proposed mechanism	5365108925	5130257296	95.62%

As we expected, the proposed method extends the time spent in the deep idle mode more than 30%. But it does not reach 100%, which is because our proposed mechanism has some intended exceptions such as RT task. We considered the requirement that RT task should finish its execution in a deadline. In this manner, we do not migrate every RT task to another CPU even the target CPU is under pseudo offline. Therefore, RT scheduler can wake up the pseudo offline CPU and reduces the time spent in the deep idle mode.

VI. CONCLUSION

This paper shows that the conventional CPU hotplug is not suitable for mobile environment because of its latency, even though hotplug is a good way to reduce power consumption. Our proposed method improves the latency by using the advanced CPU technique, deep idle mode. It is superior to the conventional hotplug in terms of power consumption. Also, our proposed method increases the time spent in the deep idle mode more than 30% compared to normal status. We believe some of these techniques may be applicable to many other user level power control algorithms for extending battery life, which we leave for future work.

ACKNOWLEDGMENT

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT Future Planning (NRF-2015M3C4A7065696). Young Ik Eom is the corresponding author of this paper.

REFERENCES

[1] A. S. V. Palladi and A. Starikovskiy, "The ondemand governor: past, present and future," in *Proceedings of Linux Symposium*, vol. 2, no. 223-238, 2001, pp. 3–3.

[2] T. Gleixner, P. E. McKenney, and V. Guittot, "Cleaning up linux's cpu hotplug for real time and energy management," *SIGBED Rev.*, vol. 9, no. 4, pp. 49–52, 2012. [Online]. Available: http://dx.doi.org/10.1145/2452537.2452547

[3] S. Brahler, "Analysis of the android architecture," *Karlsruhe institute for technology*, vol. 7, 2010.

[4] L. Brown, "ACPI in Linux," in *Linux Symposium*, vol. 51, 2005.

[5] H. Packard, Intel, Microsoft, Phoenix, and Toshiba, "Advanced configuration & power specification," July 2014. [Online]. Available: http://www.acpi.info

[6] ARM, "Idle management." [Online]. Available: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dai0143c/CHDCJAHI.html

[7] V. Guittot, "CPU hotplug." [Online]. Available: https://wiki.linaro.org/WorkingGroups/PowerManagement/Doc/Hotplug

[8] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone." in *USENIX annual technical conference*, vol. 14. Boston, MA, 2010.

[9] F. Rowand, "Using and understanding the real-time cyclictest benchmark," in *Embedded Linux Conference*, 2013.