Energy-Aware Core Switching for Mobile Devices with a Heterogeneous Multicore Processor

Junha Kim
PopcornSAR

Yeomin Nam
PopcornSAR

Moonju Park
Incheon National University

***Abstract*—Mobile devices should be designed to meet two contradictory goals: providing the performance desired by users while minimizing the energy consumption to meet the users' expectation of the battery run-time. The heterogeneous multicore architecture has been introduced to mobile world for higher energy-efficiency and performance. It is possible, however, only when the operating system assigns the appropriate core for an application to take full advantage of the heterogeneity. This article addresses the problem of the core selection for mobile devices with a heterogeneous multicore processor. It is shown that core switching for less energy consumption must consider both core frequency and utilization at the same time.**

■ **MOBILE DEVICES SHOULD** be designed to meet two contradictory goals: providing the performance desired by users while minimizing the energy consumption to meet the users' expectation of the battery run time. Balancing performance with energy is an unavoidable problem inmobile systems with limited power sources such as batteries. The heterogeneous multicore architecture (HMA) has been introduced to mobile world for higher energy-efficiency and performance. It is possible,

Digital Object Identifier 10.1109/MCE.2019.2941352
Date of current version 25 October 2019.

however, only if the operating system assigns appropriate cores for an application to take full advantage of the heterogeneity. This article focuses on the core assignment problem for mobile devices with a heterogeneous multicore processor. Real hardware is tested to see the power, the energy consumption, and the performance of the heterogeneous multicore system. It is shown that migrating applications between cores must consider both core frequency and utilization at the same time.

HETEROGENEOUS MULTICORE PROCESSORS FOR MOBILE SMART DEVICES

Mobile consumer electronics computing devices are powered by batteries that are limited in both size and capacity. For this reason, the energy efficiency has been one of the primary concerns in developing mobile devices. Recent research on the use of mobile phones has reported that the devices run out of battery almost less than half the time of the vendor-reported runtime.[1] For energy efficiency in mobile computing devices, DVFS (dynamic voltage/frequency scaling) technology is a widely used system-level tool for CPU power management. Recently, heterogeneous multicores coupling power-consuming high-performance cores with slower cores consuming less power were introduced as an architectural solution to the power management problem. An example of the HMA, called "big. LITTLE," can be found,[2] that is, currently in the market. In this architecture, there are two different types of cores in a single chip: one is called "big" designed for high performance with higher operating frequency, whereas the other core is called "LITTLE" that consumes less power but runs at slower speed. These two types of cores are different from each other in their architecture, but they share a single instruction set architecture (ISA) so that mobile systems can easily take advantage of heterogeneous multiprocessor in executing applications with different characteristics. In the operating system's viewpoint, it can provide another way of power management in addition to the DVFS technology.[3] HMA is now dominant for high-end many-core mobile systems. Mobile HMA processors appeared on the market in 2012,[4,5] and now they are widely used in high-end smart mobile devices.

In HMA, it is needed to decide on which core an application should run for energy efficiency or for performance. Because conditions and objectives of applications can be changed during run time, the best core for its execution may vary over time. In this case, the application may need to be migrated from one type of core to another; this is called "core switching."

It was pointed out[6] that the core switching cost could be high without careful analysis on performance and power usage. But, little research can be found on how core switching can be done energy-efficiently. It has been shown[7] that the CPU workload alone is not enough for effective power management. In this article, the problem of core switching in HMA for energy efficiency is discussed. Current power management techniques are reviewed, and the energy and the performance of real hardware are analyzed.

MOTIVATIONS OF HETEROGENEOUS MULTICORE FOR LESS ENERGY CONSUMPTION

Power Management With DVFS Technology

The power consumption of the CPU is divided into dynamic and static power.[8] The static power consumption is linearly proportional to the supply voltage and the leakage current. The dynamic power P dissipated by a processor can be modeled as $P \approx \alpha f V^2$ where α is a coefficient of the switching activity and the effective capacitance, f is the speed of the processor, and V is the supplying voltage. The dynamic power dissipation is the principal component in CMOS circuits. This model does not provide accurate power estimation because modern chips are not made only using CMOS. But, it provides a way to manage the power consumption by controlling the frequency and the voltage supplied to the processor.

The processor speed f is almost linearly proportional to the clock speed. When the processor is applied at a low voltage level, the frequency is lowered because of the increasing

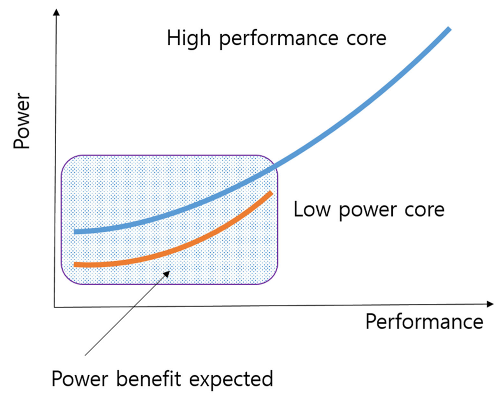

Figure 1. Power versus performance curves of HMA.

circuit delay. Consequently, the scaling of the operating frequency has a cubic impact on power dissipation of a processor. It should be noticed that the DVFS has a tradeoff between power and performance, because slowing down the CPU reduces power consumption at the expense of fewer computing cycle.[9]

Heterogeneous Multicore Processor for Low Power Consumption

Benefits of HMA are twofold: supporting diverse applications and better performance-energy efficiency.[10] As mobile devices such as smartphones require high performance with long battery run time, energy efficiency has become the primary constraint.[10,11]

Figure 1 illustrates the basic motivation of heterogeneous architecture for energy efficiency. The high-performance core has more pipeline stages and is optimized for performance, whereas the low-power core trades off performance for power efficiency. When the CPU is heavily loaded, the high-performance core can be utilized. When less CPU intensive jobs are running, the jobs can be migrated to the low-power core (i.e., core switching occurs) at the same performance point (see shaded area in Figure 1) resulting in lower power consumption without sacrificing the performance.

However, it is not obvious to implement core switching to take full advantage of the heterogeneous architecture. The performance of the core is not determined only by the operating frequency. Rather, two different cores show different performance depending on their microarchitecture. Because energy consumption is also related to the execution time of applications as well as the supply power, it is necessary to consider both power consumption and core performance all together in HMA.

FREQUENCY AND CORE MANAGEMENT IN OPERATING SYSTEMS

CPU Frequency Scaling in the Operating System

As most of the contemporary CPUs support the DVFS technology, many of the operating systems support dynamic control of CPU frequencies. Linux's *cpufreq* subsystem supports at least six kinds of CPU frequency governors that manage the frequency of the CPU according to their policies: *performance, powersave, userspace, ondemand, interactive*, and *conservative* governors.

Performance governor makes the CPU run at the highest frequency, whereas *powersave* governor sets the CPU frequency to the lowest. *Userspace* governor lets users determine the operating frequency of the CPU.

Ondemand, conservative, and *interactive* governors dynamically change the CPU frequency.

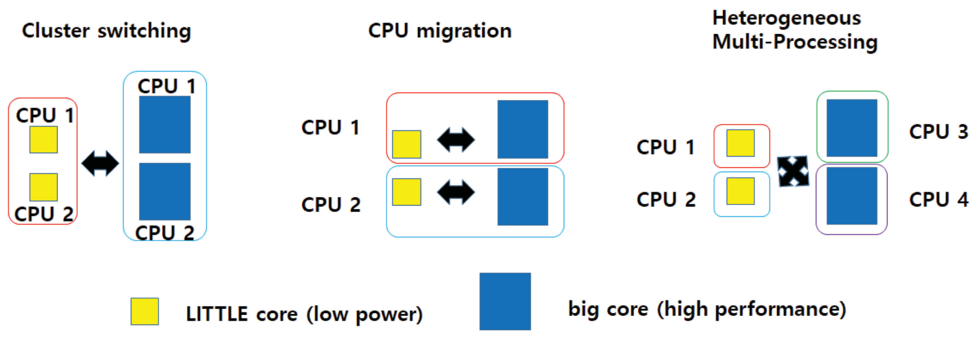

Figure 2. Core switching support for heterogeneous multicore processors.

Ondemand governor jumps up the frequency to the highest level when the CPU utilization is over the prespecified threshold. If the CPU utilization goes under the threshold, it steps down the frequency by one level. The CPU utilization is checked at a specified rate. *Conservative* governor is similar to the *ondemand* governor, but it steps up or down the frequency by one level when the processor utilization is above or below threshold values. *Interactive* governor sets the CPU frequency depending on the CPU usage similar to *ondemand* or *conservative* governor, but it speeds up the CPU more aggressively. Instead of sampling the CPU load only at a predefined rate, it checks the load after coming out of the idle state.

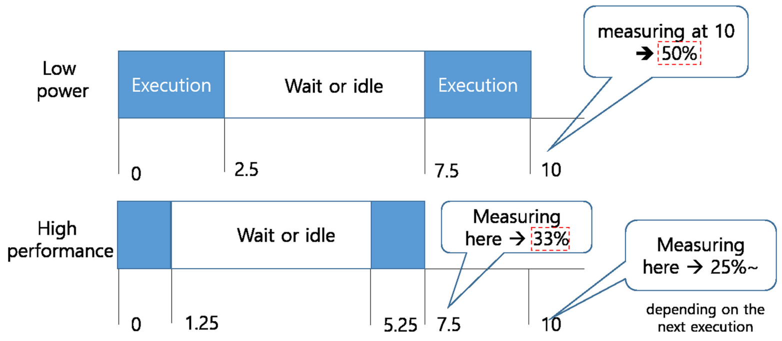

Figure 3. CPU load sampling problem in HMA.

Core Switching Scheme

Operating systems supporting HMA should determine which core an application is assigned to and which core it can be migrated to. There are three models for assigning cores to application: *cluster switching*, *CPU migration*, and *HMP* (heterogeneous multiprocessing).[4,12] Figure 2 illustrates three models of core switching. In *cluster switching*, cores of the same type are partitioned into a cluster, and only one cluster can be used at a time. *CPU migration* pairs a high-performance core with a low-power core to make a virtual core, and only one real core is activated at a time. *Cluster switching* or *CPU migration* can use only a part of the cores in a chip at a time. On the other hand, *HMP* can activate or deactivate each core individually to allow the system to utilize all cores at a time. *HMP* provides more flexibility for the system, but it makes the problem of core switching more complicated. Support for *CPU migration* was added to Linux 3.10 and core switching is done in the *cpufreq* subsystem. *HMP* is available only if the underlying hardware supports it. It has been implemented in several commercial CPUs for smartphones.[4]

Problems in CPU Load Monitoring

Most of existing schemes for frequency scaling and core switching including Linux *cpufreq* subsystem determine the CPU frequency and core migration based on the current (measured with past) CPU loads. But, HMA introduces another problem in estimating the CPU load. Figure 3 illustrates the problem with an example. In this example, it is assumed that the high-performance core has double the speed of the low-power core at the same frequency. The application runs for 2.5 time units, and waits for an I/O to be completed for 5 time units, then processes the data for 2.5 time units on the low-power core. The CPU utilization is 50% on the low-power core when it is measured at a sampling rate of 10-time unit interval.

Consider what happens if the application migrates to the high-performance core and executes the same codes. The execution time of the application will be shortened to be half, but the wait time for I/O should not be affected in theory. So, the application will finish the same job in 7.5 time units, by which the CPU is utilized about 33%, instead of 25%. But in many systems, the CPU utilization is measured at a fixed rate. Thus, the utilization can be 25% if there is no job after time 7.5, or it can be 37.5% if the application is in the loop so that it repeats the same job. This example illustrates why using the current utilization as the only parameter for the power management is not enough for HMA in designing an efficient core switching policy; it is hard to compare the loads of cores in different types. To cope with the problem, use of a weighted average of the CPU utilization has been proposed by Yu *et al.*[11] But, they did not specify how the threshold value of the CPU utilization for core switching can be determined. It can be better to define different threshold values for migration on different types of cores, which will be explained later in this article.

For experiments and analysis, a benchmark program named *cpubomb* in *Isolation Benchmark*

Table 1. Utilization of the modified benchmark program on different cores (%).

Low power (LITTLE)	10.2	20.4	29.6	40.1	50.3
High performance (big)	4.9	10.5	16.4	23.4	31.6
Low power (LITTLE)	60.5	70.6	80.0	90.5	
High performance (big)	41.3	52.0	64.6	81.5	

Suite[13] was modified to utilize only one core by a specific value. The *cpubomb* benchmark is a CPU-centric test program with a tight loop containing integer arithmetic operations, which utilizes the CPU very close to 100%. The modified benchmark tests nine levels of utilization from 10% to 90% of the low-power core at 1.3 GHz by inserting sleep times to the calculation loop. For each utilization level, fixed amount of wait times are added. This is close to real-world situation because applications have alternating sequences of CPU and I/O bursts, and the I/O wait time is typically not affected by core migration. Table 1 gives the average utilization of the benchmark program for one core measured for big and little cores at 1.3 GHz.

EXPERIMENTS: ENERGY AND PERFORMANCE ON DIFFERENT TYPES OF CORES

To see the actual effects of heterogeneous architecture, a real hardware was used for power and performance measurement. The target system has an 8-core big.LITTLE architecture mobile processor, a single-ISA HMA, described by Chung *et al.*[4] The processor has four big cores (Cortex-A15) and four LITTLE cores (Cortex-A7 cores). The system also has 2 GB LPDDR3 SDRAM at 933 MHz. A power meter with 10 Hz sampling rate has been used for measurement of power and energy consumption of the target board.

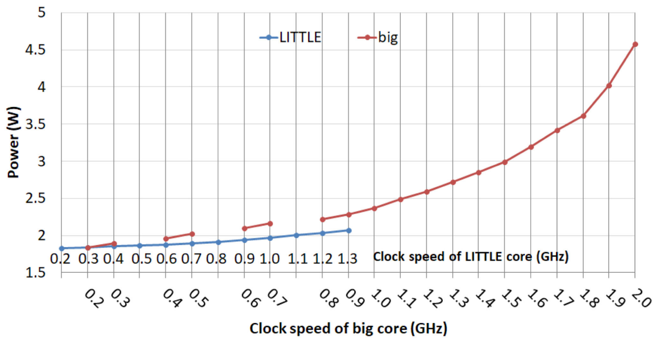

Figure 4. Power consumption at the same performance point.

Power Consumption and Performance

The average power consumption of the target board was measured to see the difference when a big core or a LITTLE core is used. When one type of core is tested, cores of the other type have been turned off. The average power consumption of the big core is always higher than that of the LITTLE core at every frequency level. However, comparing the power consumption of two different cores at the same frequency level is unfair because their performance difference is not taken into account. Considering the speedup in the big core, the power consumption graph is drawn for comparison at the same performance in Figure 4. The performance of the big core running at or over 0.9 GHz is higher than the LITTLE core at the highest speed. From Figure 4, it is evident that the LITTLE core consumes less energy with the same performance if the core is fully utilized.

Energy Consumption

The energy consumed by the modified *cpubomb* was measured with operating frequency from 0.2 GHz to 1.3 GHz on the LITTLE core and from 0.2 to 2.0 GHz on the big core. To fairly compare the energy consumption, the workload is fixed for 10 different levels from 10% to 100% of the LITTLE core utilization at 1.3 GHz. So energy consumed for the same amount of jobs is measured for each frequency level and workload, but the core utilization on a different core or at a different frequency will not be the same (except for 100% workload which does not have wait time). For example, the 90% workload which utilizes 90% of 1.3 GHz LITTLE core will utilize about 81% of 1.3 GHz big core, or 95.1% of 0.6 GHz LITTLE core. The energy consumptions on cores are compared in Figure 5. The energy consumption on big core is normalized to that on LITTLE core at the same frequency. If the normalized value goes lower than 1, it means running on big core requires less energy.

The results show that frequency range where running on big core require less energy than LITTLE core is growing as the utilization increases. As the utilization goes up, the time interval during which the core is idle gets short, resulting in the dominance of the dynamic power consumption. So, the low-power LITTLE core has the advantage when the application has more waiting time for I/O or interactions, while the high-performance big core is favorable when the CPU burst is long.

CORE SWITCHING WITH FREQUENCY SCALING

As discussed in the previous sections, it is needed to consider the performance difference in heterogeneous multicores to determine the frequency and the core assignment. But, considering both the frequency and the core assignment at the same time makes the search space too large to find a solution. Thus, two-stage approach can be taken to attack the problem. The core switching problem with cores of fixed frequency is addressed first, then its solution can be coupled with any DVFS policy available for use in the OS. This approach can also give more flexibility in choosing a DVFS policy.

In developing a core switching scheme, conditions for core switching need to be established. First, it is assumed that the core switching made only when the performance is not degraded, thus the time required to finish a job must not be delayed due to the core switching. So, core switching only occurs in two cases: when energy can be saved with the same performance, or when faster execution is possible with the same amount of energy. Second, it is assumed that the thread migration overhead is negligible.

For each workload in the experiments, the points were found at which the normalized value becomes 1 (the same amount of energy consumption) except for 10% and 100% workloads in Figure 5. The graph shows that energy can be saved by running the application on the big core if the utilization is under the value where a graph crosses $y = 1$ line. For example, consider the frequency 0.5 GHz. In this case, higher performance with lower energy can be achieved on the big core when the utilization is higher than 50%. So, it is better to run applications on a big core if the utilization is over about 72% on the LITTLE core

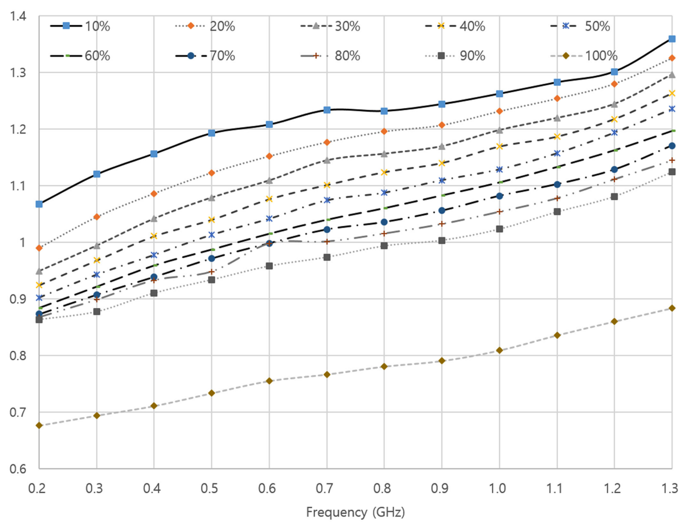

Figure 5. Energy consumption of big core execution normalized to LITTLE core execution (workload percentage is based on the LITTLE core utilization at 1.3 GHz).

(equivalent to 50% workload on big core). For the frequency level 0.8∼1.3 GHz, there is no need to switch cores for energy efficiency except 100% utilization, unless higher performance is required by a user. Table 2 summarizes the frequency and the utilization on two types of cores when the energy consumption becomes the same for each workload.

Now, consider when an application could be migrated from a big core to a LITTLE core. If an application is executing on a big core at lower than or equal to 0.8 GHz, it can be migrated to a LITTLE core without degrading performance, which results in lower energy consumption. But it should be further considered to avoid inefficient situations such that the migrated application could be sent back to the big core. To prevent this ping-pong situation, core switching should be limited to be done only when the utilization is below the values in Table 2. Table 3

Table 2. Utilization converted to different frequency levels for the same amount of the energy consumption (frequency in GHz and utilization in %).

Workload	20	30	40	50	60	70	80	90
Frequency	0.2	0.3	0.4	0.5	0.6	0.6	0.7	0.7
LITTLE	61.9	65.0	68.4	72.2	76.5	83.5	88.1	94.4
Big	41.3	43.3	45.6	48.1	51.0	55.7	58.8	62.9

Table 3. Core switching table.

Frequency (GHz)	To LITTLE utilization, target frequency	To big utilization
0.2	Below 65.0%, 0.3 GHz	Over 61.9%
0.3	Below 72.2%, 0.5 GHz	Over 65.0%
0.4	Below 76.5%, 0.6 GHz	Over 68.4%
0.5	Always, 0.8 GHz	Over 72.2%
0.6	Always, 0.9 GHz	Over 76.5%
0.7	Always, 1.1 GHz	Over 88.1%
0.8	Always, 1.2 GHz	Near 100%
0.9 and higher on big	Never	NA
0.9 ~ 1.3 on LITTLE	NA	Near 100%

summarizes the threshold values of the core utilization for core switching, and at which frequency it should be run on the migrated core. When the frequency is higher than 0.8 GHz it follows the DVFS policy of the OS.

CONCLUSION AND FUTURE DIRECTIONS

It has been shown that core switching for supporting HMA should consider both utilization and frequency. Power management techniques and core assignment policies have been reviewed, and problems in using the CPU utilization as a metric for frequency and core management were discussed. By analyzing power consumption and performance measured with a real hardware, a core switching policy can be established for a commercially available mobile multicore CPU. The suggested core switching policy can be adopted for lower energy consumption in combination with a DVFS technology.

The proposed approach can be improved further for higher energy efficiency in HMA. First, it is needed to consider the effects of on-chip caches. Performance and power consumption of modern processors are greatly affected by them. Another way for improvement can be identifying and using the characteristics of the application. Besides the CPU utilization, other factors of the application characteristics such as memory access rate and instructions per cycle can also affect the power-performance relationship. HMA provides wider range of choices for managing executions, but it also extends search space in finding an optimal solution. Extensive studies are needed to develop efficient cache management schemes, scheduling algorithms, and DVFS policies in conjunction with core assignment and switching for HMA processors.

ACKNOWLEDGMENT

This work was supported by Incheon National University Research Fund in 2016.

REFERENCES

1. T. Coughlin and IEEE Consumer Electronics Society Future Directions Committee, “A Moore’s law for mobile energy: Improving upon conventional batteries and energy sources for mobile devices,” *IEEE Consum. Electron. Mag.*, vol. 4, no. 1, pp. 74–82, Jan. 2015.
2. ARM, “big.LITTLE technology: The future of mobile,” 2013. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf
3. X. Li, G. Chen, and W. Wen, “Energy-efficient execution for repetitive app usages on big.LITTLE architectures,” in *Proc. 54th ACM/EDAC/IEEE Des. Autom. Conf.*, Austin, TX, USA, Jun. 2017, pp. 1–6.
4. H. Chung, M. Kang, and H.-D. Cho, “Heterogeneous multi-processing solution of Exynos 5 Octa with ARM big.LITTLE technology,” Samsung Electronics, Suwon, South Korea, 2012. [Online]. Available: https://www.arm.com/files/pdf/Heterogeneous_Multi_Processing_Solution_of_Exynos_5_Octa_with_ARM_bigLITTLE_Technology.pdf
5. Arm and Qualcomm, “Enabling the next mobile computing revolution with highly integrated ARMv8-A based SoCs,” 2014. [Online]. Available: https://www.arm.com/files/pdf/ARM_Qualcomm_White_paper_Final.pdf
6. S. Yoo, Y. Shim, S. Lee, S.-A. Lee, and J. Kim, “A case for bad big.LITTLE switching: how to scale power-performance in SI-HMP,” in *Proc. Workshop Power-Aware Comput. Syst.*, Oct. 2015, pp. 1–5.

7. S. Holmback, S. Lafond, and J. Lilius, “Performance monitor based power management for big.LITTLE platforms,” in *Proc. HIPEAC Workshop Energy Efficiency Heterogeneous Comput.*, Jan. 2015, pp. 1–6.
8. S. Mittal, “A survey of techniques for improving energy efficiency in embedded computing systems,” *Int. J. Comput. Aided Eng. Technol.*, vol. 6, no. 4, pp. 440–459, 2014.
9. P. Corcoran and T. Coughlin, “Safe advanced mobile power workshop,” *IEEE Consum. Electron. Mag.*, vol. 4, no. 2, pp. 10–20, Apr. 2015.
10. S. Mittal, “A survey Of techniques for architecting and managing asymmetric multicore processors,” *ACM Comput. Surv.*, vol. 28, no. 1, pp. 16–28, 2017.
11. K. Yu, D. Han, C. Youn, S. Hwang, and J. Lee, “Power-aware task scheduling for big.LITTLE mobile processor,” in *Proc. Int. SoC Des. Conf.*, Nov. 2013, pp. 208–212.
12. B. Jeff, “big.LITTLE technology moves towards fully heterogeneous global task scheduling,” ARM, 2013. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_technology_moves_towards_fully_heterogeneous_Global_Task_Scheduling.pdf
13. J. N. Matthews *et al.*, “Quantifying the performance isolation properties of virtualization systems,” in *Proc. Workshop Exp. Comput. Sci.*, Jun. 2007, Article no. 6.

Junha Kim is a Research Engineer with PopcornSAR, Inc., Seoul, South Korea. Contact him at jhkim@popcornsar.com.

Yeomin Nam is a Research Engineer with PopcornSAR, Inc., Seoul, South Korea. Contact him at ymnam@popcornsar.com.

Moonju Park is a Professor with the Department of Computer Science and Engineering, Incheon National University, Incheon, South Korea. Contact him at mpark@inu.ac.kr.