Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 18(4): 222-229, Dec. 2020

Regular paper

ETS: Efficient Task Scheduler for Per-Core DVFS Enabled Multicore Processors

Jeongkyu Hong*, ***Member*, *KIICE***

Department of Computer Engineering, Yeungnam University, Gyeongsan, Gyeongbuk 38541, Republic of Korea

Abstract

Recent multi-core processors for smart devices use per-core dynamic voltage and frequency scaling (DVFS) that enables independent voltage and frequency control of cores. However, because the conventional task scheduler was originally designed for per-core DVFS disabled processors, it cannot effectively utilize the per-core DVFS and simply allocates tasks evenly across all cores to core utilization with the same CPU frequency. Hence, we propose a novel task scheduler to effectively utilize per-core DVFS, which enables each core to have the appropriate frequency, thereby improving performance and decreasing energy consumption. The proposed scheduler classifies applications into two types, based on performance-sensitivity and allows a performance-sensitive application to have a dedicated core, which maximizes core utilization. The experimental evaluations with a real off-the-shelf smart device showed that the proposed task scheduler reduced 13.6% of CPU energy (up to 28.3%) and 3.4% of execution time (up to 24.5%) on average, as compared to the conventional task scheduler.

Index Terms: Dynamic voltage and frequency scaling, Low-power consumption, Multicore processors, Task scheduler

I. INTRODUCTION

One of the most significant problems for smart devices, such as embedded and IoT devices, is short battery life [1]. To address this problem, manufacturers have employed per-core dynamic voltage and frequency scaling (DVFS) enabled multi-core processors, which allows independent voltage and frequency control for each core. On multi-core processors with per-core DVFS, an operating system module (called a scaling governor) scales the frequency of each core individually based on core utilization. From this point of view, a task scheduler can also scale the core frequency by changing the core utilization by adjusting the amount of load allocated to each core. The task scheduler must provide an efficient task allocation method that allows the core to have the proper frequency and considers its impacts on the scaling governor.

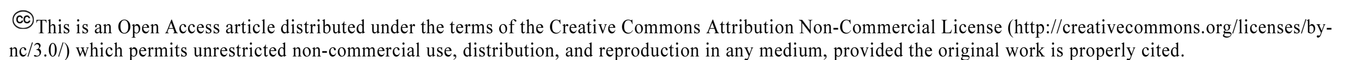
The conventional task scheduler for traditional multicore processors (that do not allow per-core DVFS) do not consider its impact on the scaling governor. Rather, it attempts to allocate the given load evenly across all cores, assuming that all cores have the same frequency. When there is a load imbalance between cores, the task scheduler migrates a task from a core with many tasks assigned to a core with fewer tasks assigned. Unfortunately, this operation disturbs the scaling governor from providing the proper frequency to cores due to the mismatch in time granularity between the task scheduler and the scaling governor.

To address the inefficiency of the conventional task scheduler, several task scheduling techniques have been proposed for smart devices with per-core DVFS-enabled multi-core processors [2, 3]. These techniques use application characteristics, such as data dependencies and user interactions, to allocate the appropriate application to cores. However, these techniques are difficult to apply to general systems, because

Received 17 November 2020, **Revised** 18 November 2020, **Accepted** 14 December 2020
***Corresponding Author** Jeongkyu Hong (E-mail: jhong@yu.ac.kr, Tel: +82-53-810-2553)
Department of Computer Engineering, Yeungnam University, Gyeongsan, Gyeongbuk 38541, Republic of Korea.

Open Access **https://doi.org/10.6109/jicce.2020.18.4.222**

print ISSN: 2234-8255 online ISSN: 2234-8883


This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Copyright Ⓒ The Korea Institute of Information and Communication Engineering

they significantly degrade the system performance for power reduction.

To overcome these limitations, and to simultaneously achieve performance improvement and energy reduction, this study proposed a novel task scheduler called ETS (efficient task scheduler). The proposed task scheduler isolates performance-sensitive applications from performance-insensitive applications, and allows performance-sensitive applications to have cores exclusively. Therefore, the scaling governor can deliver higher frequencies to the cores running performance-sensitive applications, which increases the application performance. To classify the sensitivity of the applications at runtime, ETS exploits that the child-thread configuration is different depending on the performance sensitivity of the applications.

The proposed task scheduler demonstrated the following contributions:

1) By classifying applications based on latency sensitivity, ETS reduced energy consumption by avoiding unnecessary frequency scaling.

2) By allowing performance-sensitive applications to have dedicated cores, ETS increased application performance by scaling the frequency of cores running performance-sensitive applications.

The experiments performed on a real off-the-shelf smart device showed that the proposed task scheduler achieved a 13.6% average reduction in CPU energy and a 3.4% average performance enhancement, as compared to the conventional task scheduler.

The remainder of this paper is organized as follows. Section II briefly introduces previous studies on task scheduling techniques for multi-core processors. We present the background and motivation of our work in Section III. Section IV explains our proposed task scheduler, ETS, and Section V evaluates ETS in terms of performance and CPU energy savings. Finally, the paper concludes in Section VI.

II. RELATED WORK

A. Task Scheduler for Per-Core DVFS Disabled Multicore Processors

For multi-core processors without per-core DVFS, many task scheduling techniques have been proposed. Merkel et al. tried to avoid contention for shared resources, such as cache and memory subsystems between different tasks [4]. To find the resource contention between the tasks, they estimated the resource utilization of each task. Quan and Pimentel proposed a task scheduler that considers inter-task communication costs in multi-media applications [5]. To reduce communication costs, this technique assigns tasks that communicate with each other to the same core. For real-time systems, several schemes have been proposed to assign tasks to different cores and determine the optimal core frequency based on the predefined deadline of each task [6, 7]. However, these techniques do not consider their impact on the per-core DVFS-based scaling governor. They only consider multi-core processors where all cores run at the same frequency.

B. Task Scheduler for Heterogeneous Multicore Processors

For heterogeneous multi-core processors composed of high-performance big cores and low-power small cores, the previously proposed task scheduling schemes mainly focus on the difference between big and small cores. Craeynest et al. proposed a task scheduling technique that assigns tasks with a large amount of memory level parallelism (MLP) to big cores and tasks with a large amount of instruction level parallelism (ILP) to small cores [8]. Ren et al. assigned tasks with a long execution time to the big cores and tasks with short execution time to the small cores [9]. Another task scheduling technique is proposed to consider the thermal constraints of heterogeneous multi-core processors. In the case of a thermal emergency, their technique assigns all the running tasks to the small cores to cool down the big cores quickly [10]. However, the techniques above do not consider their impact on the scaling governor for per-core DVFS.

C. Task Scheduler for Per-core Enabled Multicore Processors

For multi-core processors with per-core DVFS in smart devices, several task scheduling schemes have been proposed. Qiu et al. proposed a task scheduling technique that considers the data dependency between tasks [2]. Their technique constructs a graph that represents the dependency of tasks in an application, which determines the execution sequence of the tasks and the frequency of the cores. However, since the number of applications in the market is significantly increasing, it is not practical to construct a graph for all applications. Tseng et al. proposed another task scheduling technique that considers user attention and interaction

Table 1. Application classification

Type	Definition	Example
Performance-sensitive	Execution time is important to users	File extractor, virus scanner, and media file protector
Performance-insensitive	QoS rather than performance is important to users*	Media player, multimedia-based game, widget, and system daemon

* Insignificant applications (the applications whose load is very small) are also performance-insensitive, since their execution time is not important to users. Due to the small load, they do not require large CPU resources.

of applications [3]. This scheduling algorithm classifies applications into three levels: high (foreground applications), medium (system threads), and low (background applications), depending on the user's attention and interaction. This scheme reduces resources for less sensitive applications, while providing resources to more sensitive applications. However, this technology suffers from significant performance overhead for power reduction, which is not acceptable to users. Kim et al. proposed another task scheduler for heterogeneous multi-core processors [11]. To take advantage of the heterogeneous processors that have different-sized cores, they differentiate between multimedia and non-multimedia applications and allocate them to different cores. Because multimedia applications do not require high computing power commonly, they can save core energy consumption by mapping them to small cores.

III. BACKGROUND AND MOTIVATION

A. Application Classification

Smart device applications can be classified into two types depending on the sensitivity to the execution time, as shown in Table 1. For performance-sensitive applications, users can recognize performance degradation when the execution time increases. In contrast, for performance-insensitive applications, users only complain when the quality of service (QoS) is degraded. Because of this difference, applications with different types require different amounts of CPU resources. For performance-sensitive applications, it is appropriate to run on the core at the highest possible frequency, because the execution time typically decreases when operating at higher frequencies. Conversely, performance-insensitive applications only need the amount of CPU resources that does not degrade QoS. Hence, running performance-insensitive applications on high-frequency cores results in energy waste. In addition, different types of applications usually have different CPU utilizations. Performance-sensitive applications tend to have high CPU utilization, as they mainly consist of aperiodic and CPU-intensive operations, such as string comparison. In contrast, performance-insensitive applications have lower CPU utilization than performance-sensitive applications, because they usually consist of periodic operations. Our experiments with commercialized applications showed the average CPU utilization of performance-sensitive applications was 77.9%, significantly higher than that of performance-insensitive applications (26.8%).

B. Conventional Task Scheduler

The conventional task scheduler consists of intra- and inter-core task scheduling. The intra-core task scheduling is a completely fair scheduling (CFS), which fairly distributes CPU time slices to applications run in a core [12]. The inter-core task scheduling performs load balancing when there is a load imbalance between the cores. It migrates an application from a core with a higher load to another core with a lower load [13]. Fig. 1 shows the behavior of the conventional task scheduler as an example. We suppose application A is performance-sensitive and run on Core 0, while B1, B2, and C are performance-insensitive applications run on Core 1.

Because the CPU utilization of applications A, B, and C are different, there can be a load imbalance between Core 0 and Core 1. Hence, the load balancing migrates application C from Core 1 to Core 0 (middle of Fig. 1) to balance the load across all cores. In this case, as two different applications are run on Core 0 simultaneously, the CFS distributes the time slices to each application evenly, which results in the scaling governor adjusting the CPU frequency lower.

Unfortunately, this modulated frequency is not suitable for both applications; it is higher for performance-insensitive applications and lower for performance-sensitive applications. The time granularity of the task scheduler (0.6–7 ms [14]) is much finer than that of the scaling governor (10 ms ~ 1 s [15]) because of the overhead of voltage and frequency transition [16]. Therefore, we cannot scale the CPU frequency as much as needed. Moreover, because CFS provides limited time slices to application A, the conventional task scheduler incurs not only energy waste of application C, but also performance degradation of application A.

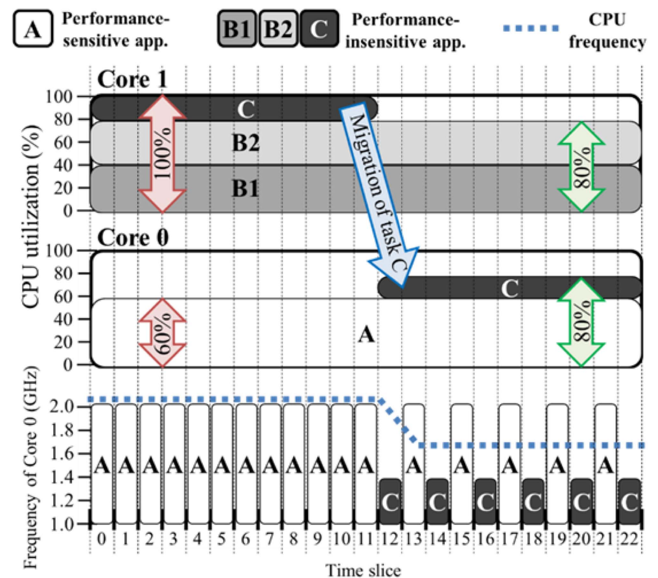


Fig. 1. Example of the conventional task scheduler. The load balancer performs the task migration (inter-core task scheduling) and the intra-core scheduling (CFS) fairly distributes time slices to all applications.

IV. ETS: EFFICIENT TASK SCHEDULING FOR PER-CORE DVFS MULTICORE PROCESSORS

To address the inefficiency of the conventional task scheduler, we proposed an efficient task scheduler called ETS, which considered its impact on the per-core DVFS-based scaling governor. Because different types of applications require different amounts of CPU resources, the proposed scheduling algorithm isolated performance-sensitive applications from performance-insensitive applications and scaled the appropriate frequency to individual cores. In addition, by providing more time slices for performance-sensitive applications, ETS reduced the execution time of performance-sensitive applications.

A. Application Type Classification

Our task scheduler classified the types of applications based on the fact that media-based applications and insignificant applications are performance-insensitive. Fig. 2 shows how ETS classifies the application types. To differentiate insignificant applications from other applications, ETS verifies how and when an application is launched. Since most insignificant applications, such as widgets and system daemons, are launched automatically and immediately after system boots-up, such applications are classified as performance-insensitive applications. In contrast, when the application is manually launched by the user, the ETS examines the child thread list of the application. As media-based applications mainly consist of periodic operations that commonly play audio tracks (e.g., music, video, and game), there is a specific thread for audio playback in the list of child threads. By exploiting this characteristic, our task scheduler classified the application as a performance-insensitive type if there was a specific thread in the list. Otherwise, the application was classified as a performance-sensitive application. Even in the case where a performance-insensitive application does not have a specific thread, our proposed ETS did not degrade its QoS because of our allocation algorithm (see details in Section IV.B.). In a real implementation, ETS classified a group of specific threads as a performance-insensitive application. Note that the grouping operation incurred negligible overhead, and thus can be performed at runtime.

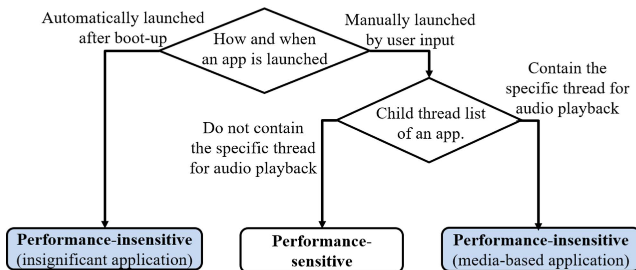


Fig. 2. Allocation type classification of ETS.

B. ETS Allocation Algorithm

Fig. 3 shows the ETS allocation algorithm for newly launched applications. If application A is launched, for example, ETS first classifies the application type through the proposed application classification algorithm (Section IV.A). After verifying the application type, ETS first checks the number of other running applications. When there is no other running application, it assigns A to core 0. Otherwise, application A is assigned to other cores depending on its type.

If application A is performance-sensitive, the proposed algorithm checked for cores that are not currently being used and then assigned the application to an unused core. In this case, application A occupies only one core. On the other hand, if there are no unused cores, the application would be assigned to a core with the lowest utilization. Because the number of running performance-sensitive applications is usually less than that of cores in smart devices [3][17], it is generally possible for performance-sensitive applications to have the cores exclusively. On the other hand, if application A is performance-insensitive, ETS would check the cores where other performance-insensitive applications are running. When there are such cores, A is allocated to the core that has the lowest utilization among the corresponding cores. Otherwise, ETS finds an unused core and allocates A to an unused core, or if there are no unused cores, it allocates A to the core with the least utilization. Even if an application type is misclassified as performance-sensitive, the QoS of performance-sensitive applications is not compromised. The proposed task scheduler allocated misclassified applications to have exclusive cores, which can lead to energy waste, but QoS was not

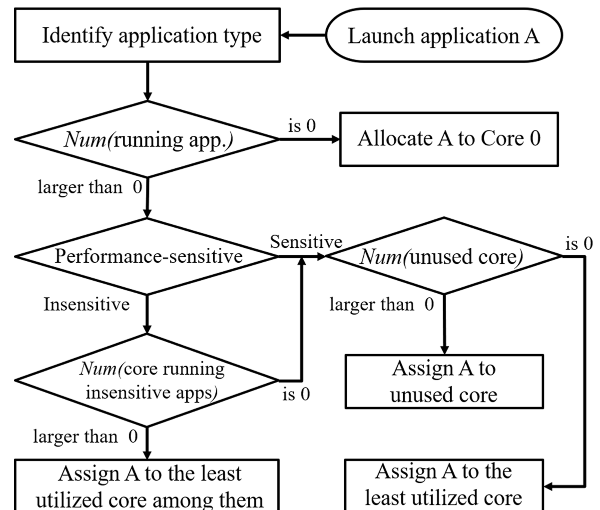


Fig. 3. Allocation algorithm of ETS.

degraded. To preserve the QoS of performance-insensitive applications, our proposed task scheduler included the load balancing algorithm for the cores where the performance-insensitive applications are running, which is discussed in the following subsection.

C. Load-Balancing Algorithm

The conventional allocation algorithm assigns performance-insensitive applications, such as media-based applications, to cores without considering QoS preservation. To preserve the QoS of the applications, our task scheduler also includes the load balancing algorithm. The proposed load-balancing algorithm is not for performance-sensitive applications, because the sensitive applications exclusively occupy cores resulting from the allocation algorithm.

The load-balancing algorithm is invoked when there is potential QoS degradation due to high utilization even in the case of the highest frequency in one of the cores where performance-insensitive applications are running. Our algorithm assumed that there was potential QoS degradation in a core when the frequency of the corresponding core reached the highest level and its utilization exceeded 80%. To maintain QoS, ETS migrated one of the performance-insensitive applications to an unused core and balanced the load across the cores where performance-insensitive applications were running. Note that because performance-insensitive applications typically have low CPU utilization, load-balancing algorithms are rarely triggered, and thus the migration overhead is negligible.

Table 2. Application combinations [11]

No.	Applications
1	Kaspersky and Amazing Alex
2	Kaspersky and Music Player
3	Kaspersky and Swampy
4	Kaspersky and MX Player
5	Gallery Lock and Amazing Alex
6	Gallery Lock and Music Player
7	Gallery Lock and Swampy
8	Gallery Lock and MX Player
9	Zipper and Amazing Alex
10	Zipper and Music Player
11	Zipper and Swampy
12	Zipper and MX Player
13	Kaspersky, Music Player, Blackbox, and 2 Widgets
14	Kaspersky, Music Player, Synthetic Performance-sensitive Application, and 2 Widgets
15	PDF Reader and Music Player
16	Google Calendar and Music Player

V. EVALUATION

A. Experimental Environment

ETS evaluation was performed on a real off-the-shelf smart device. The smart device had a per-core DVFS enabled quad-core processor [18], which supported 12 frequency steps (0.38-1.51 GHz) for each core. To employ real application cases in the evaluation, we chose 11 commercialized applications. In addition, a synthetic performance-sensitive application was selected to represent the case where multiple performance-sensitive applications are executed simultaneously, which is rare in the real world. Each application is explained in Table 2. We selected five applications (PDF Reader, Amazing Alex, Music Player, Swampy, and MX Player) based on the Moby benchmark [19]. Furthermore, six applications that have more than one million download counts were selected (Kaspersky, Gallery Lock, Zipper, Google Calendar, Blackbox, and Widget). We did not choose multiple applications from one benchmark because real use case combinations are difficult to find. In other words, users are not running Music Player and MX Player (both based on Moby benchmarks) together to listen to music.

As shown in Table 3, there are 16 practical application combinations to represent real-use cases [11]. Each combination was composed of performance-sensitive and performance-insensitive applications. In our experiments, one performance-insensitive application was executed in the foreground, while the other applications were run in the background. These combinations were also used in the previous work on the task scheduler of smart devices [3, 11]. For certain combinations that require user inputs (Nos. 15 and 16), we made a series of inputs similar to the real use cases by implementing an automatic input generator. We evaluated the proposed task scheduler using the conventional scheduler in terms of performance and energy consumption. Both task

Table 3. Application types and descriptions

Type	Name	Description
Performance-Sensitive Application	Kaspersky	Virus scanner application
	Gallery Lock	Video file protector application
	Zipper	File compressor/extractor application
	PDF Reader	PDF file reader application
	Google Calendar	Calendar application
	Synthetic Performance-sensitive Application	CPU-intensive synthetic application (for set 14 only)
Performance-Insensitive Application	Amazing Alex	Game w/ low CPU utilization
	Music Player	Music player application
	Swampy	Game w/ high CPU utilization
	MX Player	Video player application
	Blackbox	Video recorder application
	Widget	Home screen and battery widgets

schedulers used an on-demand governor as a scaling governor, which adjusted the core frequency depending on the core utilization [15]. In addition, the same DPM governor was used for all the experiments. To accurately measure energy consumption, we used an external power measuring device with a measurement error rate of 1% and ran all cases three times to ensure reliable experimental results.

B. Impact on Performance

Fig. 4 shows the execution time of performance-sensitive applications. ETS reduced the execution times by 3.4% on average and 24.5% for the best case, as compared to those of the conventional ETS. Because ETS allowed a performance-sensitive application to exclusively occupy a core, it allowed the scaling governor to provide a higher average CPU frequency to performance-sensitive applications (left side of Fig. 5). In addition, the proposed scheduler provided more time slices to performance-sensitive applications, which resulted in performance improvement.

As shown in Fig. 4, for the application sets having Kaspersky (1, 2, 3, 4, 13, and 14 sets), ETS significantly reduced the average execution time by 13.9% as compared to the conventional scheduler. As a common feature of Kaspersky is that it has many CPU-intensive operations, its performance can be enhanced by encouraging the scaling governor to provide the highest CPU frequency to the corresponding core. On the other hand, for the different application sets in Figure 4 (sets from 5–12, 15, and 16), the execution time of the proposed scheduler was slightly longer than the conventional. Unlike Kaspersky, such applications have a large I/O waiting time due to the large number of storage accesses. For example, the I/O waiting time for Gallery Locks uses up to 40%. Hence, the execution time is mainly dependent on the storage access latency, not on the CPU processing latency. Thus, ETS can barely improve performance even if it drives the scaling governor to provide higher CPU frequency to the applications.

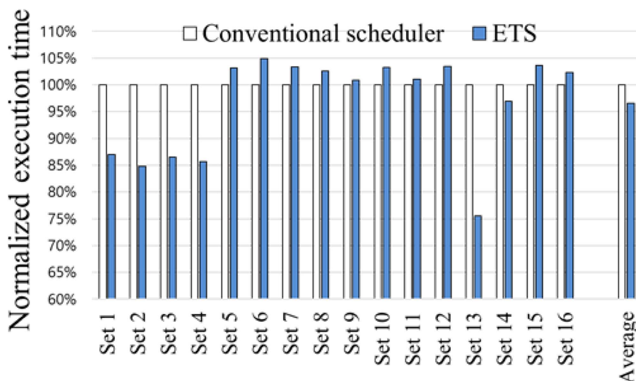


Fig. 4. Normalized execution time.

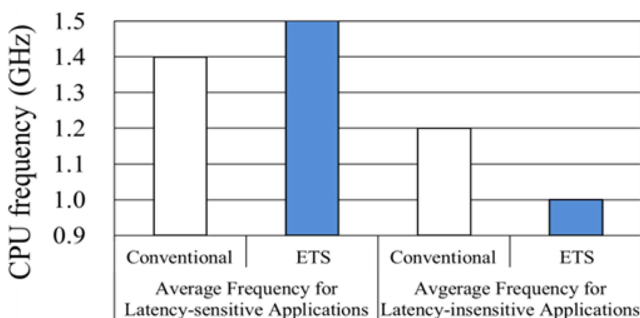


Fig. 5. CPU frequency provided to performance-sensitive applications and performance-insensitive applications.

C. Impact on Energy Consumption

Fig. 6 shows the normalized CPU energy consumption of the ETS. Because there is no way to accurately measure CPU energy consumption in our experimental environment, CPU energy consumption was estimated from the total system energy, assuming that it accounted for 30% [17, 20]. As shown in Fig. 6, ETS reduces CPU energy consumption by 13.6% on average (up to 28.3%) compared to the conventional task scheduler, because it separates performance-insensitive applications from performance-sensitive applications. Thus, allowing the scaling governor to provide a lower average frequency of CPUs running performance-insensitive applications.

For application sets having Gallery Lock, Zipper, and interactive applications (i.e., sets 5-12, 15, and 16), ETS saved energy mainly due to CPU power reduction. For these sets, although the execution time using ETS was slightly longer than the conventional scheduling algorithm as explained before, the proposed scheduler significantly saved the CPU power consumption (15.9% on average) by preventing the scaling governor from providing unnecessarily high frequency to cores running performance-insensitive applications. On the other hand, for application sets including Kaspersky (sets

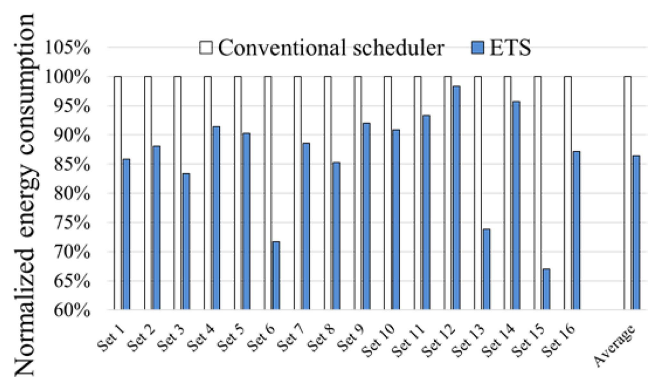


Fig. 6. Normalized CPU energy consumption.

1-4, 13, and 14), ETS reduced energy consumption mainly through performance enhancement. Since ETS provided the highest frequency to the core running Kaspersky, it did not significantly reduce CPU power consumption. However, it was possible to achieve a 13.6% average energy reduction by significantly improving the execution time of Kaspersky, as explained in Section VI.A.

VI. CONCLUSIONS

Recent smart devices adopt multi-core processors that allow per-core DVFS, which enables per-core voltage and frequency scaling. However, the conventional task scheduler does not consider its impact on the per-core DVFS-based scaling governor and may provide inappropriate CPU frequency to the cores, which can degrade application performance and increase energy waste. To address this problem, we proposed an efficient task scheduler called ETS that considered the operations of the scaling governor. ETS categorized application types according to performance sensitivity, allowing performance-sensitive applications to have exclusive cores, and scaling CPU frequencies as needed to achieve optimal performance. Because the classification was performed by referring to the child thread list of each application, it can be done without noticeable overheads at runtime. In addition, ETS reduced energy waste by avoiding unnecessary frequency scaling for performance-insensitive applications performed by the scaling governors. Compared to the conventional task scheduler, the experimental results showed that the proposed task scheduler reduced the CPU energy by 13.6% (up to 28.3%), as well as the execution time by 3.4% (up to 24.5%) on average.

ACKNOWLEDGEMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1G1A1003780).

REFERENCES

[1] M. Kim, Y. G. Kim, and S. W. Chung, "Measuring variance between smartphone energy consumption and battery life," *IEEE Computer*, vol. 47, no. 7, pp. 59-65, 2014. DOI: 10.1109/MC.2013.293.

[2] M. Qiu, Z. Chen, L. T. Yang, X. Qin, and B. Wang, "Towards power-efficient smartphones by energy-aware dynamic task scheduling," in *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*, Liverpool, pp. 1466-1472, 2012. DOI: 10.1109/HPCC.2012.214.

[3] P. Tseng, P. Hsiu, C. Pan, and T. Kuo, "User-centric energy-efficient scheduling on multi-core mobile devices," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, CA, pp. 1-6, 2014. DOI: 10.1109/DAC.2014.6881412.

[4] A. Merkel, J. Stoess, and F. Bellosa, "Resource-conscious scheduling for energy efficiency on multicore processors," in *Proceedings of the 5th European Conference on Computer Systems (EuroSys '10). Association for Computing Machinery*, New York: NY, USA, pp. 153-166, 2010. DOI:10.1145/1755913.1755930.

[5] W. Quan and A. D. Pimentel, "A scenario-based run-time task mapping algorithm for MPSoCs," in *Proceedings of the 50th Annual Design Automation Conference (DAC '13). Association for Computing Machinery,* New York: NY, USA, pp. 1-6, 2013. DOI: 10.1145/2463209.2488895.

[6] L. Niu, "System-level energy-efficient scheduling for hard real-time embedded systems," *Design, Automation & Test in Europe*, Grenoble, pp. 1-4, 2011. DOI: 10.1109/DATE.2011.5763275.

[7] E. Seo, J. Jeong, S. Park, and J. Lee, "Energy efficient scheduling of real-time tasks on multicore processors," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1540-1552, 2008. DOI: 10.1109/TPDS.2008.104.

[8] K. V. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, Portland, OR, pp. 213-224, 2012. DOI: 10.1109/ISCA.2012.6237019.

[9] S. Ren, Y. He, S. Elnikety, and K. McKinley, "Exploiting processor heterogeneity for interactive services," In *Proceedings of International Conference on Autonomic Computing (ICAC '13)*, pp. 45-58, 2013. DOI: 10.1145/3123939.3123956.

[10] Y. G. Kim, M. Kim, J. M. Kim, and S. W. Chung, "M-DTM: Migration-based dynamic thermal management for heterogeneous mobile multi-core processors," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, pp. 1533-1538, 2015.

[11] Y. G. Kim, M. Kim, and S. W. Chung, "Enhancing energy efficiency of multimedia applications in heterogeneous mobile multi-core processors," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1878-1889, 2017. DOI: 10.1109/TC.2017.2710317.

[12] C. S. Pabla, "Completely fair scheduler," *Linux Journal*, vol. 184, no. 4, pp. 1, 2009.

[13] P. U. Murthy, "Overview of the current approaches to enhance the linux scheduler," *Linux Foundation Collaboration Summit,* 2013.

[14] P. Chubb, "Linux, Locking and Lots of Processors," 2018. [online] Available: https://www.cse.unsw.edu.au/~cs9242/13/lectures/06-scalability.pdf.

[15] D. Brodowski and N. Golde, "Linux CPUFreq governors," [online] Available: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[16] W. Kim, M. S. Gupta, G. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *IEEE 14th International Symposium on High Performance Computer Architecture*, Salt Lake City, UT, pp. 123-134, 2008. DOI: 10.1109/HPCA.2008.4658633.

[17] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with Eprof," in *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys '12). Association for Computing Machinery*, New York: NY, USA, pp. 29-42, 2012. DOI: 10.1145/2168836.2168841.

[18] Qualcomm. Snapdragon S4 Pro. [online] Available: http://www.qualcomm.com/snapdragon/processors/s4/specs

[19] Y. Huang, Z. Zha, M. Chen, and L. Zhang, "Moby: A mobile benchmark suite for architectural simulators," in *IEEE International Symposium on Performance Analysis of Systems and Software*

(ISPASS), Monterey, CA, pp. 45-54, 2014. DOI: 10.1109/ISPASS.2014.6844460.

[20] A. Shye, B. Scholbrock, and G. Memik, “Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42). Association for Computing Machinery,* New York: NY, USA, pp. 168-178, 2009. DOI: 10.1145/1669112.1669135.

Jeongkyu Hong

He received the B.S. degree from the College of Information and Communications, Korea University, in 2011; the M.S. degree from the Department of Computer Science, Korea Advanced Institute of Science and Technology (KAIST), in 2013; and the Ph.D. degree from the School of Computing, KAIST, in 2017. He has been with the Department of Computer Engineering, Yeungnam University, since 2018, as an Associate Professor. His current research interests include the design of low power, reliable, and high-performance processor and memory systems.