# Enhancing Energy Efficiency of Multimedia Applications in Heterogeneous Mobile Multi-Core Processors

Young Geun Kim, Minyong Kim, *Student Member, IEEE*, and Sung Woo Chung, *Senior Member, IEEE*

**Abstract**—Recent smart devices have adopted heterogeneous multi-core processors which have high-performance big cores and low-power small cores. Unfortunately, the conventional task scheduler for heterogeneous multi-core processors does not provide appropriate amount of CPU resources for multimedia applications (whose QoS is important to users), resulting in energy waste; it often executes multimedia applications and non-multimedia applications on the same core. In this paper, we propose an advanced task scheduler for heterogeneous multi-core processors, which provides appropriate amount of CPU resources for multimedia applications. Our proposed task scheduler isolates multimedia applications from non-multimedia applications at runtime, exploiting the fact that multimedia applications have a specific thread for video/audio playback (to play video/audio, a multimedia application should use a function that generates the specific thread). Since multimedia applications usually require a smaller amount of CPU resources than non-multimedia applications due to dedicated hardware decoders, our proposed task scheduler allocates the former to the small cores and the latter to the big cores. In our experiments on an Android-based development board, our proposed task scheduler saves system-wide (not just CPU) energy consumption by 8.9 percent, on average, compared to the conventional task scheduler, preserving QoS of multimedia applications. In addition, it improves performance of non-multimedia applications by 13.7 percent, on average, compared to the conventional task scheduler.

**Index Terms**—Multimedia application, energy management, task scheduler, heterogeneous mobile multi-core processor, smart device

## 1 Introduction

With a rapid growth of mobile market, the number of smart device users has been continuously increased [39]. To improve mobile experience of the increased number of users, recent smart devices (e.g., smartphones, tablets, etc.) have employed a wide range of computing resources such as hardware encoder/decoder [5], [6]. Thanks to the computing resources, users have enjoyed sophisticated multimedia applications [40] (e.g., game applications, video player applications, music player applications, etc.) without heavy computation on CPU.

For multimedia applications, users are sensitive to QoS (Quality of Service). Due to this reason, preserving QoS of multimedia applications is one of the challenging problems in smart devices. Unfortunately, it is not desirable to provide as large amount of CPU resources as possible for multimedia applications in smart devices, since smart devices are energy constrained [19]. Therefore, it is necessary to provide appropriate amount of CPU resources for multimedia applications, which preserves QoS while minimizing energy consumption [1]; though multimedia applications usually do not require a large amount of CPU resources due to the dedicated hardware decoders, they still require CPU resources, since CPU deals with input/output of hardware decoders.

Recent smart devices have adopted heterogeneous multi-core processors such as ARM's big.LITTLE [10], which have high-performance big cores and low-power small cores. To utilize different types of cores in heterogeneous multi-core processors, the conventional task scheduler migrates applications between different types of cores depending on CPU utilization of each application [44]. When CPU utilization of an application on a small core exceeds a pre-defined threshold, the conventional task scheduler migrates the application to a big core. On the other hand, when CPU utilization of an application on a big core decreases below another pre-defined threshold, it migrates the application to a small core.

Unfortunately, such conventional task scheduler often executes multimedia applications and non-multimedia applications (e.g., virus scanner applications, file compressor/extractor applications, etc.) on the same core. In this case, CPU frequency scaling governor[1] cannot provide appropriate frequency for multimedia applications and higher frequency for non-multimedia applications due to context switches between them. For example, when a context switch occurs from a non-multimedia application to a multimedia application, it provides inappropriate frequency for the

1. CPU frequency scaling governor is an operating system module which periodically measures utilization of each CPU core and dynamically adjusts CPU frequency as well as voltage depending on the measured utilization [26].

• *The authors are with the Department of Computer Science, Korea University, Seoul 136-713, Korea. E-mail: {carrotyone, mkim05, swchung}@korea.ac.kr.*

*Manuscript received 25 July 2016; revised 1 May 2017; accepted 23 May 2017. Date of publication 31 May 2017; date of current version 16 Oct. 2017. (Corresponding author: Sung Woo Chung.)*
*Recommended for acceptance by D. Marculescu.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TC.2017.2710317*

multimedia application; it adjusts frequency depending on CPU utilization of the non-multimedia application (the example is shown in Fig. 2 of Section 3.2). In addition, the conventional task scheduler cannot provide enough cache and CPU time slices for both of applications; multimedia applications should compete with non-multimedia applications for cache and CPU time slices. For these reasons, the conventional task scheduler causes energy waste of multimedia applications as well as performance degradation of non-multimedia applications.

In this paper, we propose an advanced task scheduler for heterogeneous mobile multi-core processors, which enhances energy efficiency of multimedia applications. Our proposed task scheduler isolates multimedia applications from non-multimedia applications at runtime, exploiting the fact that multimedia applications have a specific thread for video/audio playback (to play video/audio, a multimedia application should use a function which generates the specific thread). Since multimedia applications generally require a smaller amount of CPU resources than non-multimedia applications due to the dedicated hardware decoders [5], [6], our proposed task scheduler allocates the former to the small cores and the latter to the big cores[2]. In this way, our proposed task scheduler brings the following two advantages at the same time:

1) Energy saving: the proposed technique provides appropriately small amount of CPU resources for multimedia applications, preserving QoS
2) Performance improvement: the proposed technique provides larger amount of CPU resources for non-multimedia applications.

For evaluation, we implement our proposed task scheduler on an Android-based development board.

The rest of this paper is organized as follows. In Section 2, we introduce previous studies on energy management techniques for multi-core processors. In Section 3, we present background of our work. In Section 4, we explain our proposed task scheduler. In Section 5, we evaluate our proposed task scheduler in terms of system-wide energy consumption, QoS, and performance. Finally, we conclude our work in Section 6.

## 2 Related Work

To save CPU energy consumption for general applications, many techniques have been proposed based on DVFS (Dynamic Voltage and Frequency Scaling) or task scheduling. Generally, DVFS-based techniques adjust CPU voltage and frequency depending on resource demand of applications [14], [21]. To find appropriate CPU voltage and frequency for the applications, DVFS-based techniques often utilize feedback-based control algorithms [2], [9]. Generally, the feedback-based control algorithms estimate performance and power consumption of applications, using sophisticated application models. After that, they scale CPU voltage and frequency to minimize the gap between the estimated values and the target values. On the other hand, task scheduling-based techniques usually allocate (or migrate) applications to CPU cores, considering their impact on CPU power states (voltage/frequency or the number of active cores) [20], [35]. For heterogeneous multi-core processors in servers or desktops, several task scheduling techniques have been proposed to efficiently utilize different types of cores. Craeynest et al. proposed a task scheduling technique that allocates tasks to different types of cores depending on ILP (Instruction Level Parallelism) and MLP (Memory Level Parallelism) [7]. Petrucci et al. proposed another task scheduling technique for heterogeneous multi-core processors, which allocates CPU-intensive applications to the big cores and memory-intensive applications to the small cores [29]. Ren et al. tried to allocate the applications with long execution time to the big cores and the applications with short execution time to the small cores [32]. Integrating DVFS and task scheduling, EAS (Energy Aware Scheduling) was introduced in Linux [43]. The EAS periodically measures CPU utilization of each application. Based on the measured utilization, it estimates power and performance of applications on all the CPU cores. After that, the EAS allocates (or migrates) each application to a core which is likely to consume the smallest amount of power while minimizing performance loss. Liu et al. also tried to integrate DVFS and task scheduling in order to maximize performance of applications while maintaining CPU power consumption under the pre-defined power budget [23]. At first, their proposed technique allocates threads to high-performance cores and scales voltage/frequency to the highest values. After that, it migrates some threads to the low-power cores and gradually reduces voltage/frequency of the cores, until the estimated CPU power consumption is below the power budget. Stamoulis and Marculescu proposed another similar DVFS and task scheduling technique that considers process variations across the CPU cores [36]. Their proposed technique includes a CPU power model that takes into account the processes variations. Based on the model, it allocates threads to CPU cores and scales voltage/frequency of the cores, which satisfies performance constraints of applications while maintaining CPU power consumption under the pre-defined power budget. Based on [33], several techniques integrated DVFS and task scheduling to maintain power consumption of multi-core processors under the power budget [16], [34], considering temperature of CPU cores. The techniques include thermal models which estimate temperature of CPU cores based on CPU power states. Using the thermal model, the techniques find the highest CPU power state that keeps temperature under the predefined threshold; leakage power of CPU cores is highly dependent on temperature. In addition, the techniques determine the location of active cores considering heat transfer across the cores and allocate threads to the active cores. Khdr et al. further tried to maintain not only power consumption of CPU cores but also that of accelerators under the predefined power budgets based on DVFS as well as task scheduling [17]. However, all the above techniques cannot provide

2. Our proposed task scheduler treats all child treads of an application as one application; in mobile OSs, an application is usually composed of one or more child threads and each thread is represented as a task. For this reason, it allocates (or migrates) all child threads of an application to a core. Note, according to our analysis on top 100 applications of Google Play Store, allocating (or migrating) all child threads to a core does not cause any problem due to following two reasons: 1) in an application, only one child thread is executed during much of entire execution time (63.7 percent, on average) and 2) the average number of simultaneously running threads in an application is not large (1.5, on average).

appropriate amount of CPU resources for multimedia applications in smart devices, since they do not consider features of multimedia applications whose QoS is important to users; multimedia applications usually require a smaller amount of CPU resources than non-multimedia applications due to the dedicated hardware decoders [5], [6].

To enhance energy efficiency of multimedia applications in smart devices, many DVFS-based techniques have been proposed, considering features of multimedia applications. Chang et al. proposed a DVFS-based technique that considers resource usage patterns of multimedia applications [4]. Their proposed technique constructs a state machine for each multimedia application, where each state corresponds to resource combination (e.g., memory and external storage, memory and display, etc.) used by the application. At runtime, their proposed technique maintains history of state transitions and average CPU utilization. Based on the history, their proposed technique predicts the next state and adjusts CPU frequency to the lowest frequency that is likely to keep the CPU utilization below a certain value in the next state. Das et al. proposed another DVFS-based technique for multimedia applications [8]. In the offline phase, their proposed technique classifies multimedia applications into a fixed set of classes depending on CPU utilization. In addition, their proposed technique finds the lowest CPU frequency for each applications class, which does not harm QoS. In the online phase, their proposed technique adjusts CPU frequency according to classes of running multimedia applications. Khan et al. also tried to adjust frequency of CPU cores for multimedia applications [15]. Their proposed technique includes a CPU power model which estimates CPU power consumption based on the frequency. By using the power model, their proposed technique finds the highest CPU frequency that keeps CPU power consumption below the pre-defined power budget. Since CPU, GPU, and several IP (Intellectual Property) cores (e.g., decoder) usually cooperate in multimedia applications, several techniques attempt to adjust frequency of GPU and IP cores along with that of CPU, considering performance slack of multimedia applications [25], [27], [30]. However, the above techniques do not consider impact of task scheduling on CPU power states. In addition, they do not consider heterogeneous multi-core processors which have different types of cores with different performance.

Several techniques have been proposed based on task scheduling, targeting energy reduction of multimedia applications. For homogeneous multi-core processors which have only one type of cores, Quan and Pimentel proposed a task scheduling technique that considers communication costs between tasks in a multimedia application [31]. Their proposed technique includes a graph whose edges correspond to communication costs between the tasks in a multimedia application. By using the graph, their proposed technique determines an execution order of the tasks, which minimizes communication costs between the tasks. However, the technique does not consider different types of cores with different performance in heterogeneous multi-core processors. Note big cores are usually faster than small cores due to higher frequencies, larger number of pipeline stages, and larger cache sizes; for an XML parsing benchmark (BaseMark OS II Suite), a big core shows 42.0 percent higher performance than a small core while consuming 62.0 percent higher energy in Samsung Exynos 7420 [24]; only in a few cases, the big cores show lower performance, compared to the small cores [13], [18]. For heterogeneous multi-core processors, Hsiu et al. proposed a task scheduling technique that considers user attention and interaction of applications [11]. Their proposed technique classifies applications into three sensitivity levels depending on user attention and interaction: high (foreground applications), medium (system threads), and low (background applications). Their proposed technique provides a smaller amount of CPU resources for the applications with lower sensitivity to provide a larger amount of CPU resources for the applications with higher sensitivity. However, their proposed technique significantly degrades performance of background applications (67.6 percent, on average) for energy saving, which may not be acceptable to users; users might wait for completion of background applications while doing other things. Pathania et al. proposed another task scheduling technique which allocates threads in a multimedia application to different types of cores, considering their impact on FPS [28]. In case that a thread has a large impact on FPS due to high CPU utilization, their proposed technique allocates the thread to the big cores. Otherwise, it allocates the thread to the small cores. After allocating threads, their technique scales the CPU frequency to the lowest that keeps FPS above a certain value. However, the technique does not consider non-multimedia applications which also have large impact on energy efficiency of multimedia applications; users often execute both multimedia and non-multimedia applications at the same time in recent smart devices [37]. Different from the above techniques, our proposed task scheduler provides appropriate amount of CPU resources for multimedia applications and non-multimedia applications at the same time (details are explained in Section 4).

TABLE 1
Application Classification in Smart Devices

| Application Type | Definition | Example |
|---|---|---|
| Multimedia Application | Applications whose QoS is important to users | Music player, video player, and game based on multimedia |
| Non-multimedia Application | Applications whose execution time is important to users | Virus scanner, file extractor, and PDF reader |
| Insignificant Application | Applications whose CPU utilization is very low | Widget |

## 3 Background

### 3.1 Applications for Smart Devices

In smart devices, applications are classified into three types, as shown in Table 1. For multimedia applications which commonly play video/audio, users are sensitive to QoS. On the other hand, for non-multimedia applications, users notice performance degradation, when execution time gets longer. For insignificant applications, both QoS and execution time are not so important to users.

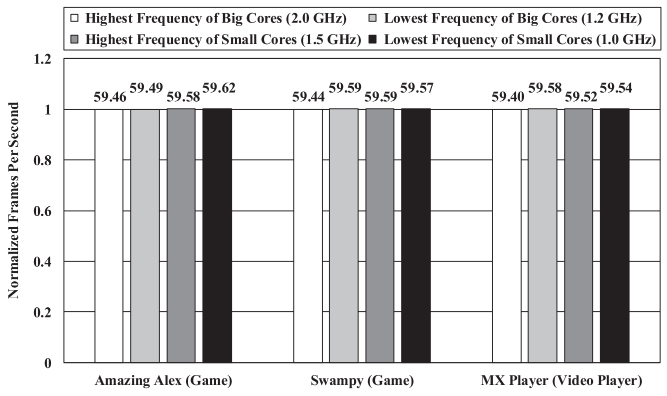


Fig. 1. Normalized FPS (Frames Per Second) of multimedia applications at different CPU frequencies. (Note FPS is normalized to that at the highest frequency of big cores.)

Due to different application features, different types of applications require different amount of CPU resources (e.g., CPU frequency, CPU time slice, and cache); since users often execute multiple foreground applications at the same time in recent smart devices (e.g., iPad Pro and Galaxy Note) [37], it becomes more important to provide appropriate amount of CPU resources for different applications. In the past, researchers thought multimedia applications required a large amount of CPU resources to maintain QoS. However, this is not true any longer in recent smart devices. Since recent smart devices have employed dedicated hardware decoders for multimedia applications[3] [5], [6], multimedia applications usually do not require a large amount of CPU resources for QoS preservation; in our experiments on an Android-based development board with heterogeneous multi-core processors [41], QoS of multimedia applications is usually not degraded even at the lowest small core frequency, as shown in Fig. 1.[4] Thus, executing multimedia applications with an unnecessarily large amount of CPU resources only wastes energy. On the other hand, non-multimedia applications require as large amount of CPU resources as possible, since execution time usually decreases with a larger amount of CPU resources. Insignificant applications do not require a large amount of CPU resources, since they usually have low CPU utilization. Note in our experiments with real world applications, average CPU utilization of non-multimedia applications (57.9 percent) was much higher than that of multimedia applications (10.2 percent) and insignificant applications (less than 5.0 percent); CPU utilization of each application was measured at the highest frequency of the big cores.

### 3.2 Inefficiency of Conventional Task Scheduler for Multimedia Applications

The conventional task scheduler currently used for heterogeneous mobile multi-core processors consists of Global Task Scheduling (GTS), load balancing, and Completely Fair Scheduling (CFS). Each of them operates as follows:

1) *GTS* allocates (or migrates) applications between different types of cores, depending on CPU utilization of each application [44]. Whenever an application is launched, GTS allocates the newly launched application to an unused big core and measures CPU utilization of the application on the big core. As long as the measured CPU utilization is higher than a predefined threshold, *hmp_down_threshold* (25.0 percent, in our experiments), the GTS executes the application on the core. Only when the CPU utilization is lower than *hmp_down_threshold*, GTS migrates the application to an active small core which has the lowest CPU utilization; note, in off-the-shelf smart devices, such as Galaxy S6, the GTS tries to allocate as many applications to a single small core as possible to reduce the number of active small cores. GTS executes the application on the small core, as long as the measured CPU utilization is lower than another threshold, *hmp_up_threshold* (68.4 percent, in our experiments). Only when the CPU utilization exceeds *hmp_up_threshold*, GTS migrates the application to an unused big core.
2) *Load balancing* balances CPU utilization in each type of cores. Specifically, it migrates an application from a core with larger utilization to another active core with smaller utilization [38].
3) *CFS* distributes CPU time slices to applications as fairly as possible within a core [38].

Fig. 2 shows an example of the conventional task scheduler (to simplify the example, we assume that only one big core and one small core are active). Suppose a user plays a game based on multimedia (a multimedia application) in the foreground while executing a file extractor (a non-multimedia application) in the background.

When the file extractor performs CPU-intensive operations to extract data from a compressed file, GTS of the conventional task scheduler executes the file extractor on a big core

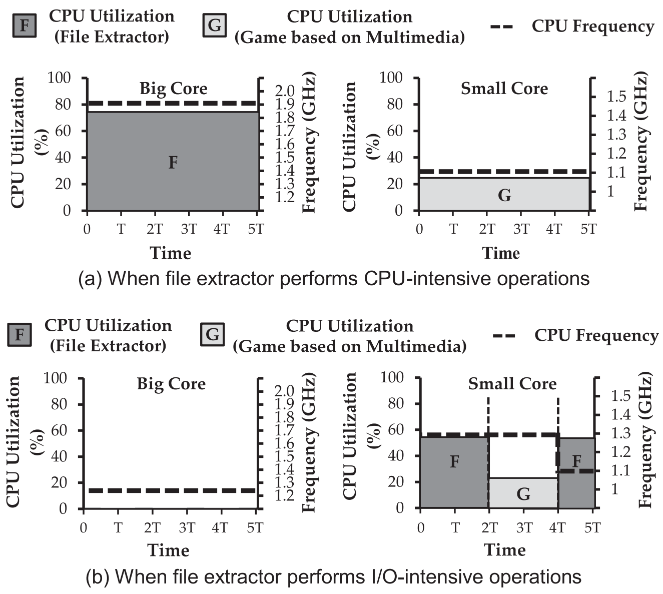


Fig. 2. Example of conventional task scheduler.

3. Recent smart devices have employed dedicated hardware decoders to preserve QoS with less power consumption. Note, only in case that the hardware decoders do not support a format of video/audio, software decoding is inevitably used. However, such a case rarely occurs in real world multimedia applications.

4. For Amazing Alex and Swampy, an Android service (Surface Flinger) limits FPS to the display refresh rate (60 FPS in our development board) [27]. For MX Player, we played a video file that has 60 FPS.

due to high CPU utilization, as shown in Fig. 2a. On the contrary, GTS executes the game on a small core due to low CPU utilization. In this case, since only the game runs on the small core, CPU frequency scaling governor delivers appropriate frequency to the game, as shown in the right side of Fig. 2a. In addition, it provides sufficiently high frequency for the file extractor, as shown in the left side of Fig. 2a.

On the other hand, when the file extractor performs I/O-intensive operations to load the compressed file from external storage, CPU utilization of the file extractor decreases. Due to the low CPU utilization on the big core, GTS migrates the file extractor from the big core to the small core, as shown in Fig. 2b; despite I/O operations, CPU utilization of file extractor is still higher than that of the game at the same frequency of the small core. Since the game and file extractor run on a small core together, CFS of the conventional task scheduler distributes CPU time slices to the applications, as shown in the right side of Fig. 2b. Unfortunately, in this case, CPU frequency scaling governor fails to provide appropriate CPU frequency for the game as well as the file extractor due to context switches between them. For example, at time 2T in the right side of Fig. 2b, a context switch occurs from the file extractor to the game. In this case, CPU frequency scaling governor provides unnecessarily high frequency for the game, since it scales CPU frequency depending on the previous CPU utilization (CPU utilization of the file extractor). Similarly, after time 4T in the right side of Fig. 2b, it provides low frequency for the file extractor. Note time granularity of CPU frequency scaling governor is usually coarser than that of context switch. Consequently, the conventional task scheduler causes energy waste of the multimedia application (the game) as well as performance degradation of the non-multimedia application (the file extractor).

To eliminate the above inefficiency of the conventional task scheduler, we propose an advanced task scheduler for heterogeneous mobile multi-core processors. Our proposed task scheduler allocates multimedia applications to the small cores and non-multimedia applications to the big cores; the former usually require a smaller amount of CPU resources than the latter, as explained in Section 3.1. By isolating multimedia applications from non-multimedia applications, CPU frequency scaling governor provides appropriate small core frequency for multimedia applications and higher big core frequency for non-multimedia applications, as shown in Fig. 2a; it adjusts small core and big core frequency depending on CPU utilization of multimedia and non-multimedia applications, respectively. Additionally, our proposed task scheduler prevents multimedia applications from competing with non-multimedia applications for cache and CPU time slices. In the next section, we explain details of our proposed scheduler.

## 4 An Energy-Efficient Task Scheduler for Multimedia Applications

In this section, we propose an advanced task scheduler for heterogeneous mobile multi-core processors,[5] which enhances energy efficiency of multimedia applications. As shown in Fig. 3, the proposed task scheduler classifies application types at runtime by using a light-weight application type classification method (Section 4.1). Based on the classified application types, our proposed task scheduler allocates multimedia applications to the small cores and non-multimedia applications to the big cores (Section 4.2). After allocating applications, our proposed task scheduler migrates multimedia applications from the small cores to the big cores, when there is no small core that preserves QoS (Section 4.3). Even if the type of an application is mis-classified (or changed at runtime), our proposed task scheduler does not degrade QoS or performance of the application (Section 4.4).

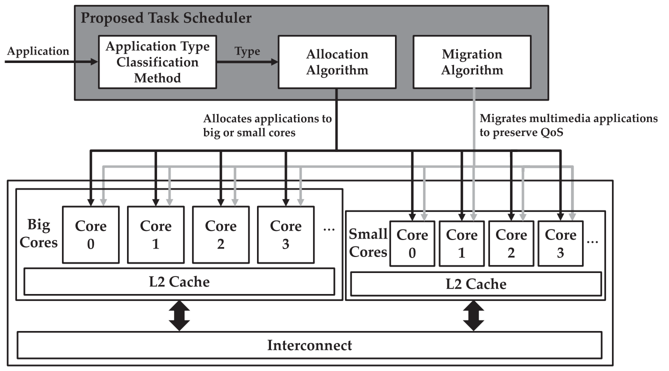


Fig. 3. Overview of our proposed task scheduler.

### 4.1 Application Type Classification Method

Our proposed task scheduler differentiates multimedia applications from non-multimedia applications at runtime, since the former usually requires smaller amount of CPU resources than the latter due to the dedicated hardware decoders [5], [6]. In addition, our proposed task scheduler differentiates insignificant applications (whose CPU load is very small) from other applications, since insignificant applications also do not require a large amount of CPU resources.

Fig. 4 shows the application type classification method of our proposed task scheduler. To differentiate multimedia applications from non-multimedia applications, the method first investigates the child thread list of an application. In case that the method finds out a specific thread for video/audio playback from the list, it classifies the application as a multimedia application (multimedia applications have the specific thread for video/audio playback in their child thread list, while the others do not). Note in our implementation on an Android OS,[6] we use *AudioTrack* as the specific thread; to play video/audio, an Android-based application should use the *AudioTrack* API (Application Programming Interface) which generates the *AudioTrack* thread.

5. Our proposed task scheduler allocates (or migrates) OS services (e.g., Surface Flinger, location manager service, Blink/Webkit rendering engine, etc.) to different types of cores depending on their CPU utilization, in order to preserve performance of the OS services; it is feasible to differentiate the OS services from other user-level applications, since there are only limited number of OS services in mobile OSs.

6. Though we implement our proposed task scheduler on an Android OS, the application classification method of our proposed task scheduler is also applicable to the other OSs; mobile application developers usually use given APIs for audio/video playback (e.g., *AVAudioPlayer* in iOS and *sound-player* in Tizen), which generate the specific child threads.

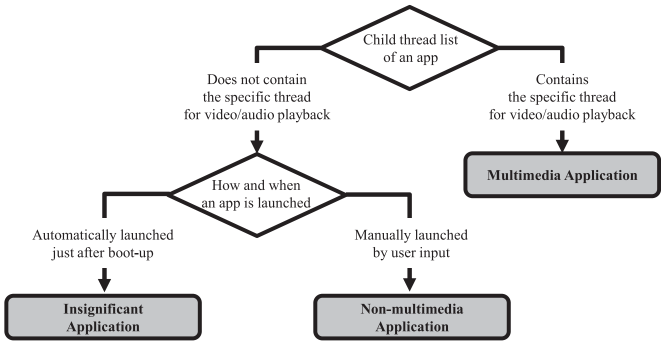

Child thread list of an app
Does not contain the specific thread for video/audio playback
Contains the specific thread for video/audio playback
Multimedia Application
How and when an app is launched
Automatically launched just after boot-up
Manually launched by user input
Insignificant Application
Non-multimedia Application

Fig. 4. Application type classification method of our proposed task scheduler.

In case that the specific thread is not included in the list, the method examines how and when the application is launched. If the application is manually launched by a user input, the method classifies the application as a non-multimedia application; users might be sensitive to completion of the application, since they manually launched it. On the other hand, if the application is automatically launched just after boot-up, the method classifies the application as an insignificant application; most of the insignificant applications (e.g., widgets) are automatically launched just after boot-up. Since the child thread investigation and launch time detection incur negligible overhead, the method classifies the types of applications without affecting QoS or performance.

Note, for top 100 applications on Google Play Store, the method achieves 97 percent accuracy; it only fails when a multimedia application uses a self-developed API for audio/video playback, which does not generate the specific thread. Even if the application type classification method mis-classifies the type of an application, our proposed task scheduler does not harm QoS or performance of the application (details are explained in Section 4.4).

## 4.2 Allocation Algorithm

Fig. 5 describes the allocation algorithm of our proposed task scheduler. When an application $\alpha$ is launched, the allocation algorithm classifies the type of $\alpha$ by using the application type classification method (explained in Section 4.1). After classifying the type, it allocates $\alpha$ to the big or small cores depending on the type.

When $\alpha$ is a multimedia application, the allocation algorithm searches for an unused core[7] among the small cores. In case that an unused small core exists, it allocates $\alpha$ to the core. Otherwise, it allocates $\alpha$ to a small core which has the lowest CPU utilization.

When $\alpha$ is not a multimedia application but a non-multimedia application, the allocation algorithm searches for an unused core among the big cores. In case that an unused big core exists, it allocates $\alpha$ to the core. Otherwise, it allocates $\alpha$ to a big core which has the lowest CPU utilization. Note it is

7. If there are a large number of big and small cores, searching for an unused core may degrade performance of our proposed task scheduler. To solve the problem, our proposed task scheduler can further adopt other searching algorithms, such as randomized strategies [3].

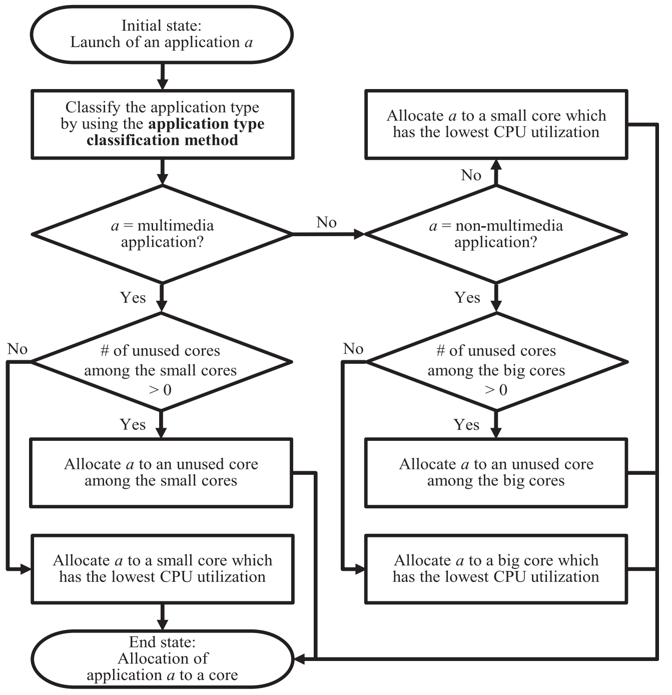

Initial state: Launch of an application a
Classify the application type by using the application type classification method
a = multimedia application?
No
a = non-multimedia application?
No
Allocate a to a small core which has the lowest CPU utilization
Yes
Yes
No
# of unused cores among the small cores > 0
No
# of unused cores among the big cores > 0
Yes
Yes
Allocate a to an unused core among the small cores
Allocate a to an unused core among the big cores
Allocate a to a small core which has the lowest CPU utilization
Allocate a to a big core which has the lowest CPU utilization
End state: Allocation of application a to a core

Fig. 5. Allocation algorithm of our proposed task scheduler.

possible to exclusively allocate one non-multimedia application to a big core in most cases, leading to performance improvement; the number of running non-multimedia applications (not threads) is usually less than that of big cores in a real world [11], [37].

When $\alpha$ is an insignificant application, the allocation algorithm allocates $\alpha$ to a small core which has the lowest CPU utilization, since insignificant applications do not require a large amount of CPU resources.

By isolating multimedia applications from non-multimedia applications, the allocation algorithm provides QoS preserving smallest amount of CPU resources for the former and larger amount of CPU resources for the latter, as explained in Section 3.2.

## 4.3 Migration Algorithm

The allocation algorithm allocates multimedia applications to the small cores, since they usually have low CPU utilization due to the dedicated hardware decoders [5], [6]. However, when the hardware decoders do not support a format of video/audio, software decoding is inevitably used. In this case, QoS of the applications might not be satisfied on the small cores, since CPU utilization of the applications increases due to the software decoding. To preserve QoS in such a case, our proposed task scheduler includes a migration algorithm.

Fig. 6 describes the migration algorithm of our proposed task scheduler. At the beginning, the migration algorithm checks CPU frequency and utilization of the small cores where multimedia applications run. In case that there is a small core whose CPU frequency is the highest frequency and CPU utilization exceeds a predefined threshold (*up_threshold*), the migration algorithm checks the number of applications running on the small core. When multiple applications are running on the core, our proposed task scheduler migrates one of the applications

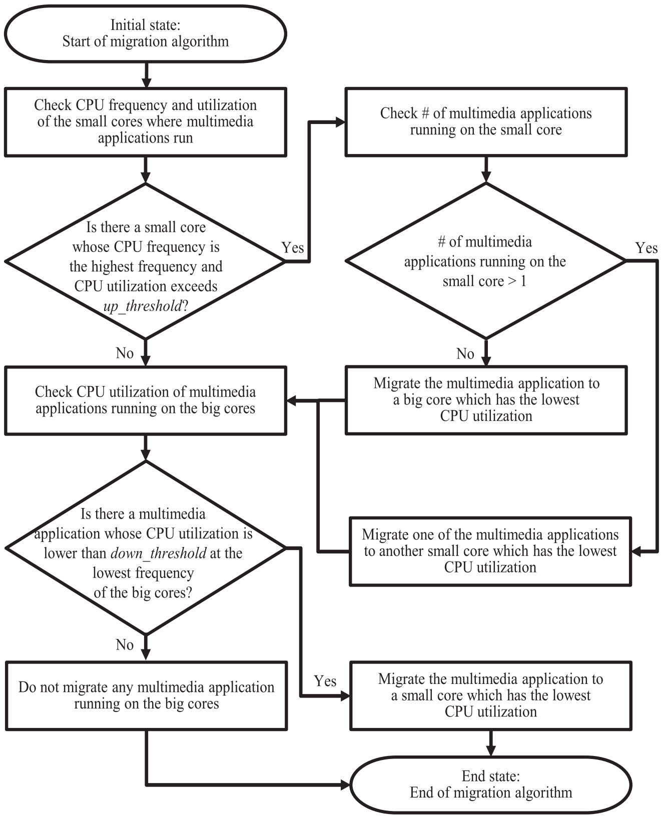

Initial state:
Start of migration algorithm
Check CPU frequency and utilization of the small cores where multimedia applications run
Is there a small core whose CPU frequency is the highest frequency and CPU utilization exceeds up_threshold?
Yes
No
Check # of multimedia applications running on the small core
# of multimedia applications running on the small core > 1
Yes
No
Migrate the multimedia application to a big core which has the lowest CPU utilization
Migrate one of the multimedia applications to another small core which has the lowest CPU utilization
Check CPU utilization of multimedia applications running on the big cores
Is there a multimedia application whose CPU utilization is lower than down_threshold at the lowest frequency of the big cores?
No
Yes
Do not migrate any multimedia application running on the big cores
Migrate the multimedia application to a small core which has the lowest CPU utilization
End state:
End of migration algorithm

Fig. 6. Migration algorithm of the proposed task scheduler for multimedia applications.

to another small core which has the lowest CPU utilization. On the other hand, when only one application is running on the core, the algorithm migrates the application to a big core, since QoS of the application is not satisfied on the small core.

The migration algorithm also checks CPU utilization of multimedia applications running on the big cores. When there is a multimedia application whose CPU utilization is lower than another threshold (*down_threshold*) at the lowest frequency of the big cores, the migration algorithm migrates the application to a small core which has the lowest CPU utilization. On the other hand, when there is no multimedia application whose CPU utilization is lower than *down_threshold* at the lowest frequency of the big cores, the migration algorithm does not migrate any multimedia application running on the big cores.

In our implementation on an Android OS, we use 80 percent and 20 percent for *up_threshold* and *down_threshold*, respectively.[8] After experiments with various real world application sets, we confirm that any QoS degradation does not occur in our proposed task scheduler (the detailed results are explained in Section 5.2.2). In addition, the migration algorithm rarely migrates multimedia applications across the cores, since they usually have low CPU utilization due to the dedicated hardware decoders.

Note the migration algorithm is not for non-multimedia applications, since a non-multimedia application exclusively occupies one big core, as explained in Section 4.2. In addition, it does not consider insignificant applications whose QoS is not so important to users.

### 4.4 Robustness to Application Type Mis-Classification

Even if the type of an application is mis-classified (or changed at runtime), our proposed task scheduler does not degrade QoS or performance of the application. In this subsection, we present how our proposed task scheduler handles mis-classified applications.

In case that the type of a multimedia application is mis-classified as a non-multimedia application (a multimedia application does not have a specific thread for video/audio playback), the allocation algorithm of our proposed task scheduler exclusively allocates the mis-classified application to a big core, as explained in Section 4.2. In this case, the allocation algorithm provides larger amount of CPU resources for the mis-classified application, which causes additional energy consumption, still maintaining QoS; note the additional energy consumption is marginal (1.1 percent, on average, for multimedia applications used in our experiments), since the CPU frequency scaling governor reduces big core frequency depending on CPU utilization. However, such a case rarely occurs in a real world, since most multimedia applications use the specific thread to play video/audio, as explained in Section 4.1. Similarly, even if the type of a non-multimedia application is changed to a multimedia application at runtime, QoS is not hurt; the application exclusively occupies one big core, since the migration algorithm does not migrate non-multimedia applications between different types of cores. Note, when a non-multimedia application, such as web browser, plays video using an internal movie player, its type is changed to a multimedia application.

On the other hand, in case that the type of a non-multimedia application is mis-classified as a multimedia application (a non-multimedia application has a specific thread for video/audio playback), the allocation algorithm of our proposed task scheduler first allocates the mis-classified application to a small core. However, as soon as CPU utilization of the mis-classified application exceeds *up_threshold* at the highest small core frequency, the migration algorithm migrates the application to a big core, as explained in Section 4.3. Due to this reason, our proposed task scheduler does not degrade performance of the mis-classified application, compared to the conventional task scheduler. In addition, even if the type of a multimedia application is changed to a non-multimedia application at runtime, performance is not degraded; the migration algorithm migrates the application depending on CPU utilization. Note, however, it is difficult to find such a case in a real world.

## 5 Evaluation

### 5.1 Experimental Environment

We perform our evaluation on Odroid-XU3 [41], a development board running Android version 4.4.2 (KitKat) and

8. Though the threshold values are defined through a sensitivity test, they can be changed to reflect up-to-date characteristics of real world application sets. In our experiments, we found these values are not crucial in most real world application sets, since four small cores are enough for QoS preservation of all the running multimedia applications; multimedia applications usually have low CPU utilization due to the dedicated hardware decoders.

TABLE 2
Description of Applications

| Classification | Name | Description | Average CPU Utilization[a] |
|---|---|---|---|
| Multimedia Application | Amazing Alex | Game w/ low CPU utilization | 8.7% |
| | Music Player | Music player application | 5.3% |
| | Swampy | Game w/ low CPU utilization | 21.6% |
| | MX Player | Video player application | 10.2% |
| | Blackbox | Video recorder application | 5.4% |
| | Edge of Tomorrow | Game w/high CPU utilization | 82.0% |
| Non-multimedia Application | Kaspersky | Virus scanner application | 92.5% |
| | Gallery Lock | Video file protector application | 54.9% |
| | Zipper | File compressor and extractor application | 31.9% |
| | Kingsoft Office | DOC, PPT, and XLS file reader application | 55.6% |
| | PDF Reader | PDF file reader application | 51.0% |
| | Google Calendar | Calendar application | 32.5% |
| | Synthetic Non-multimedia Application | CPU-intensive synthetic application (for set 17 only) | 87.6% |
| Insignificant Application | Widget | Home screen widget and battery widget | 1.2% |

[a]. *Average CPU utilization of each application is measured at the highest frequency of the big core.*

TABLE 3
Application Sets

| Set No. | Applications |
|---|---|
| 1 | Kaspersky and Amazing Alex |
| 2 | Kaspersky and Music Player |
| 3 | Kaspersky and Swampy |
| 4 | Kaspersky and MX Player |
| 5 | Kaspersky and Edge of Tomorrow |
| 6 | Gallery Lock and Amazing Alex |
| 7 | Gallery Lock and Music Player |
| 8 | Gallery Lock and Swampy |
| 9 | Gallery Lock and MX Player |
| 10 | Gallery Lock and Edge of Tomorrow |
| 11 | Zipper and Amazing Alex |
| 12 | Zipper and Music Player |
| 13 | Zipper and Swampy |
| 14 | Zipper and MX Player |
| 15 | Zipper and Edge of Tomorrow |
| 16 | Kaspersky, Music Player, Blackbox, and 2 Widgets |
| 17 | Kaspersky, Music Player, Synthetic Non-multimedia Application, and 2 Widgets |
| 18 | Kingsoft Office and Music Player |
| 19 | PDF Reader and Music Player |
| 20 | Google Calendar and Music Player |

Linux kernel 3.10. Odroid-XU3 has Samsung Exynos 5,422 [46], which runs four high-performance cores (big cores) and four low-power cores (small cores) at the same time. The processor supports 9 frequency steps (from 2,000 to 1,200 MHz) and 6 frequency steps (from 1,500 to 1,000 MHz) for the big cores and small cores, respectively. Note, on Odroid-XU3, all cores in the same type of cores operate at the same frequency.

To explore real use cases in our evaluation, we select 13 real world applications reflecting the opinion of smart device industries, as shown in Table 2. We also select a synthetic non-multimedia application for the case where multiple non-multimedia applications run simultaneously; it is difficult to find such a case in a real world. We select 6 applications (Kingsoft Office, PDF Reader, Amazing Alex, Music Player, Swampy, and MX Player) referring to Moby benchmark [12]. In addition, we select 6 applications (Kaspersky, Gallery Lock, Zipper, Google Calendar, Blackbox, and Widget) whose download counts exceed 1,000,000. We also select one application (Edge of Tomorrow) referring to [27], in order to confirm that our proposed task scheduler does not harm QoS of CPU-intensive multimedia applications. The reason why we do not select applications from one benchmark is that it is difficult to find real use case combinations. For example, in a real use case, it does not make sense to run Amazing Alex (game) and Swampy (game) together, since users usually cannot enjoy two games simultaneously.

For our evaluation, we have 20 practical application sets which represent real use cases, as shown in Table 3. The application sets are combinations of multimedia applications, non-multimedia applications, and insignificant applications. In case of application sets including Amazing Alex, Swampy, MX Player, and Edge of Tomorrow, we execute one multimedia application in the foreground, while running the others in the background. On the other hand, in case of the application sets including Music Player, we execute one non-multimedia application in the foreground, while running the others in the background. Note these kinds of combinations were also used in recent previous works on the task scheduler of smart devices [11], [37]. For application sets including user interaction (set 18, 19, and 20), we use an automatic input generator to mimic a series of inputs just like the real use cases.

We compare our proposed task scheduler (denoted as $TS_{proposed}$) to the conventional task scheduler (denoted as $TS_{conventional}$) in terms of system-wide energy consumption, QoS of multimedia applications, and performance of non-multimedia applications; note TS stands for Task Scheduler. For both of the task schedulers, we use interactive governor as a CPU frequency scaling governor, which scales frequency of each type of cores depending on its utilization [42]. We measure system-wide energy consumption using Monsoon Power Meter [45], an accurate power measurement device with 1 percent measurement error. Due to the small measurement error, many previous works have used the same device to perform energy evaluation on real smart devices [11], [22], [37]. For QoS evaluation, we measure FPS of multimedia applications (except for Music Player) referring to the frame logs of an Android service (Surface Flinger), which updates display frame buffer [27]. To confirm that the experimental results are consistent, we run

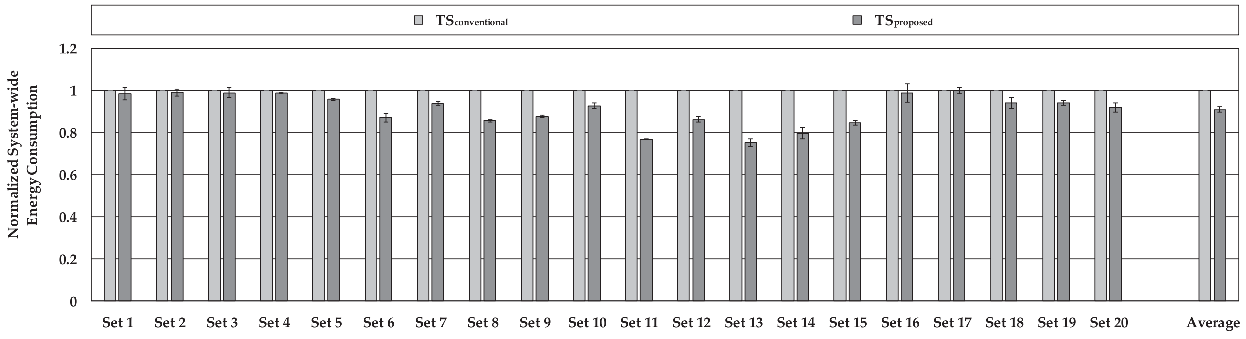


Fig. 7. Normalized system-wide energy consumption. (Note system-wide energy consumption of $TS_{proposed}$ is normalized to that of $TS_{conventional}$.)

every case three times; the average measurement difference among the three runs is 1.9 percent.

## 5.2 Experimental Result

### 5.2.1 System-Wide Energy Consumption

Fig. 7 shows system-wide energy consumption of $TS_{proposed}$ normalized to that of $TS_{conventional}$. Note the error bar represents standard deviation of the three runs. As shown in Fig. 7, $TS_{proposed}$ saves average system-wide (not just CPU) energy consumption by 8.9 percent (24.6 percent in the best case), compared to $TS_{conventional}$. There are two reasons:

1) $TS_{proposed}$ does not execute non-multimedia applications on the small cores where multimedia applications run. Due to the reason, the CPU frequency scaling governor provides lower small core frequency for multimedia applications, as shown in the left side of Fig. 8.
2) $TS_{proposed}$ executes only one non-multimedia application on a big core. Due to the reason, the CPU frequency scaling governor provides higher big core frequency for non-multimedia applications, as shown in the right side of Fig. 8, improving execution time.

For the application sets (set 6 ~ 15 and 18 ~ 20 in Fig. 7) including the non-multimedia applications (Gallery Lock, Zipper, Kingsoft Office, PDF Reader, and Google Calendar), $TS_{proposed}$ saves more system-wide energy consumption (13.0 percent, on average, compared to $TS_{conventional}$), compared to the other application sets. The common feature of the non-multimedia applications (Gallery Lock, Zipper, Kingsoft Office, PDF Reader, and Google Calendar) is that they have a large amount of I/O operations. During I/O operations, CPU utilization of the non-multimedia applications decreases. Due to the low CPU utilization on the big cores, $TS_{conventional}$ often migrates the non-multimedia applications to the small cores. In this case, $TS_{conventional}$ causes following two problems: 1) CPU frequency scaling governor provides unnecessarily high frequency for multimedia applications, since it scales CPU frequency depending on CPU utilization of non-multimedia applications; despite I/O operations, at the same frequency, average CPU utilization of the non-multimedia applications (50.5 percent) is still much higher than that of the multimedia applications (26.4 percent) and 2) CPU frequency scaling governor provides low small core frequency for non-multimedia applications, degrading execution time. Different from $TS_{conventional}$, $TS_{proposed}$ prevents CPU frequency scaling governor from unnecessarily raising small core frequency for multimedia applications by isolating them from the non-multimedia applications. In addition, $TS_{proposed}$ significantly improves execution time of the non-multimedia applications, as it allocates a non-multimedia application to a big core exclusively; the CPU frequency scaling governor provides sufficiently high big core frequency for non-multimedia applications. For these reasons, $TS_{proposed}$ significantly saves system-wide energy consumption for these application sets.

In case of the application sets (set 1 ~ 5 and 16 in Fig. 7) including the non-multimedia application (Kaspersky), system-wide energy consumption of $TS_{proposed}$ is similar to that of $TS_{conventional}$. Different from other non-multimedia applications, Kaspersky has a large amount of CPU-intensive operations. In this case, due to high CPU utilization of Kaspersky, $TS_{conventional}$ does not frequently migrate Kaspersky to the small cores where multimedia applications run. For this reason, though $TS_{proposed}$ isolates multimedia applications from Kaspersky, it does not reduce much energy consumption, compared to $TS_{conventional}$. Similarly, in case of the applications set (set 17 in Fig. 7) including multiple non-multimedia applications (Kaspersky and Synthetic Non-multimedia Application), system-wide energy consumption of $TS_{proposed}$ is similar to that of $TS_{conventional}$. The reason is that both $TS_{proposed}$ and $TS_{conventional}$ execute the non-multimedia applications much time on the big cores due to high CPU utilization.

Note, though $TS_{proposed}$ migrates Edge of Tomorrow between the big and small cores, it still saves system-wide energy consumption of the application sets (set 5, 10, and 15 in Fig. 7) including Edge of Tomorrow. The reason is that $TS_{proposed}$ improves execution time of the non-multimedia applications, since it exclusively allocates a non-multimedia application to a big core.

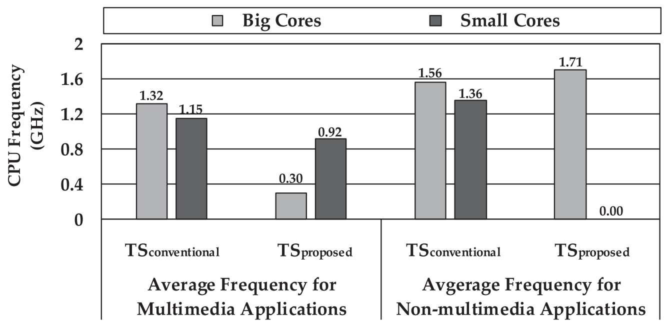


Fig. 8. Average CPU frequency provided to multimedia applications and non-multimedia applications.

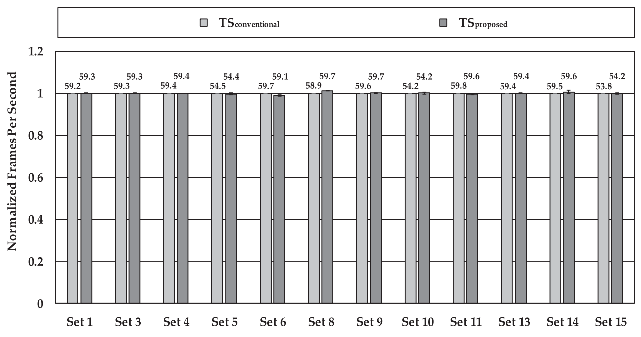


Fig. 9. Normalized FPS (Frames Per Second) of multimedia applications. (Note FPS of $TS_{proposed}$ is normalized to that of $TS_{conventional}$.)

### 5.2.2 QoS of Multimedia Applications

Fig. 9 shows FPS of multimedia applications except for Music Player (for Music Player, we confirmed that users did not perceive any difference between the two task schedulers through a user study). As shown in Fig. 9, FPS of $TS_{proposed}$ is similar to that of $TS_{conventional}$.

In case of the application sets including Amazing Alex, Swampy, and MX Player (set 1 $\sim$ 4, 6 $\sim$ 9, and 11 $\sim$ 14 in Fig. 9), $TS_{proposed}$ isolates multimedia applications from non-multimedia applications, by allocating the former to the small cores and the latter to the big cores. In this case, CPU frequency scaling governor provides appropriate small core frequency for the multimedia applications, preserving QoS. In addition, $TS_{proposed}$ prevents the multimedia applications from competing with the non-multimedia applications for cache and CPU time slices. As a result, $TS_{proposed}$ preserves QoS of the multimedia applications; both of task schedulers almost achieve the highest achievable FPS for these application sets (the highest achievable FPS of each multimedia application is explained in Section 3.1).

In case of the application sets including Edge of Tomorrow (set 5, 10, and 15 in Fig. 9), $TS_{proposed}$ does not harm QoS of Edge of Tomorrow, compared to $TS_{conventional}$. The reason is that both $TS_{proposed}$ and $TS_{conventional}$ migrate Edge of Tomorrow between the big and small cores, depending on its CPU utilization.

### 5.2.3 Performance of Non-Multimedia Applications

Fig. 10 shows execution time of non-multimedia applications. As shown in Fig. 10, $TS_{proposed}$ improves execution time of the applications by 13.7 percent, on average (31.9 percent in the best case), compared to $TS_{conventional}$. Since $TS_{proposed}$ exclusively allocates one non-multimedia application to a big core, CPU frequency scaling governor provides higher big core frequency for the non-multimedia applications, as shown in the right side of Fig. 8. Moreover, $TS_{proposed}$ makes the non-multimedia applications exclusively occupy cache and CPU time slices. As a result, $TS_{proposed}$ improves execution time of non-multimedia applications.

For the application sets (set 6 $\sim$ 15 and 18 $\sim$ 20 in Fig. 10) including the non-multimedia applications (Gallery Lock, Zipper, Kingsoft Office, PDF Reader, and Google Calendar), $TS_{proposed}$ improves execution time more significantly than the other application sets (20.4 percent, on average, compared to $TS_{conventional}$). In this case, $TS_{proposed}$ provides a larger amount of CPU resources (CPU frequency, CPU time slices, and cache) for the non-multimedia applications (Gallery Lock, Zipper, Kingsoft Office, PDF Reader, and Google Calendar), compared to $TS_{conventional}$; it always executes the applications on the big cores, while $TS_{conventional}$ migrates the applications between the big and small cores depending on CPU utilization. In addition, $TS_{proposed}$ prevents the applications from frequently migrating between the big and small cores, reducing migration overhead. Due to these reasons, $TS_{proposed}$ significantly improves execution time of the applications.

In case of the application sets (set 1 $\sim$ 5 and 16 $\sim$ 17 in Fig. 10) including the non-multimedia applications (Kaspersky as well as Synthetic Non-multimedia Application), execution time of $TS_{proposed}$ is similar to that of $TS_{conventional}$. In this case, since both $TS_{proposed}$ and $TS_{conventional}$ execute Kaspersky and Synthetic Non-multimedia Application most time on the big cores due to high CPU utilization, $TS_{proposed}$ does not reduce much execution time, compared to $TS_{conventional}$.

## 6 Conclusion

Recent smart devices have adopted heterogeneous mobile multi-core processors which have different types of cores with different performance. In heterogeneous mobile multi-core processors, the conventional task scheduler migrates applications between different types of cores depending on CPU utilization of each application. Unfortunately, the conventional task scheduler makes it difficult for CPU frequency scaling governor to provide appropriate frequency for multimedia applications (whose QoS is important to users), resulting in energy waste; it often executes multimedia applications and non-multimedia applications on the same core.

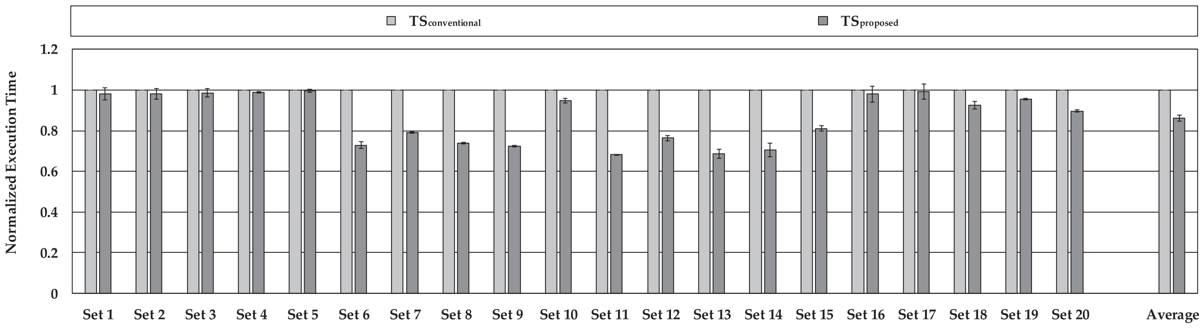


Fig. 10. Normalized execution time of non-multimedia applications. (Note execution time of $TS_{proposed}$ is normalized to that of $TS_{conventional}$.)

In this paper, we propose an advanced task scheduler for heterogeneous mobile multi-core processors, which enhances energy efficiency of multimedia applications. Our proposed task scheduler differentiates multimedia applications from non-multimedia applications at runtime, exploiting the fact that multimedia applications have a specific thread for video/audio playback. Since multimedia applications usually require a smaller amount of CPU resources than non-multimedia applications due to the dedicated hardware decoders, our proposed task scheduler allocates former to the small cores and the latter to the big cores. By isolating multimedia applications from non-multimedia applications, our proposed task scheduler helps CPU frequency scaling governor to provide appropriate CPU frequency for multimedia applications and higher frequency for non-multimedia applications. In addition, it prevents multimedia applications from competing with non-multimedia applications for cache and CPU time slices. In our experiments on an Android-based development board, our task scheduler saves system-wide (not just CPU) energy consumption by 8.9 percent, on average, compared to the conventional task scheduler, preserving QoS of multimedia applications. In addition, it improves performance of non-multimedia applications by 13.7 percent, on average, compared to the conventional task scheduler. We believe that our proposed task scheduler will be adopted in off-the-shelf smart devices to enhance energy efficiency of multimedia applications while preserving QoS.

## ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2015R1A2A1A15055435 and No. 2014R1A2A1A11054390) and Samsung Electronics. We would also like to thank the editor and anonymous reviewers for their helpful feedback.

## REFERENCES

[1] S. Ahmed and B. H. Ferri, "Prediction-based asynchronous CPU-budget allocation for soft-real-time applications," *IEEE Trans. Comput.*, vol. 63, no. 9, pp. 2343–2355, Sep. 2014.

[2] P. Bogdan, R. Marculescu, and S. Jain, "Dynamic power management for system-on-chip platforms: An optimal control approach," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 4, article 46, pp. 1–20, 2013.

[3] P. Bogdan, T. Sauerwald, A. Stauffer, and H. Sun, "Balls into bins via local search," in *Proc. ACM-SIAM Symp. Discrete Algorithms*, 2013, pp. 16–34.

[4] Y.-M. Chang, P.-C. Hsiu, Y.-H. Chang, and C.-W. Chang, "A resource-driven DVFS scheme for smart handheld devices," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 3, article 54, pp. 1–22, 2013.

[5] T. Chen, A. Rucker, and G. E. Suh, "Execution time prediction for energy-efficient hardware accelerators," in *Proc. 48th Int. Symp. Microarchitecture*, 2015, pp. 457–469.

[6] T. M. Chen, L.-J. Lin, H.-L. Chen, K.-C. Liu, C.-L. Su, and C.-Y. Cho, "A DVB-T implementation for Android Stagefright on a heterogeneous multi-core platform," in *Proc. 14th Int. Conf. High Performance Comput. Commun. 9th Int. Conf. Embedded Softw. Syst.*, 2012, pp. 112–118.

[7] K. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-core through performance impact estimation (PIE)," in *Proc. Int. Symp. Comput. Archit.*, 2012, pp. 213–224.

[8] A. Das, A. Kumar, B. Veeravalli, R. Shafik, G. Merrett, and B. Al-Hashimi, "Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems," in *Proc. Des. Autom. Test Europe Conf.*, 2015, pp. 43–48.

[9] R. David, P. Bogdan, and R. Marculescu, "Dynamic power management for multicores: Case study using the intel SCC," in *Proc. Int. Conf. VLSI Syst.-on-Chip*, 2012, pp. 147–152.

[10] P. Greenhalgh, "big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," *ARM White Paper*, pp. 1–8, 2011.

[11] P.-C. Hsiu, P.-H. Tseng, W.-M. Chen, C.-C. Pan, and T.-W. Kuo, "User-centric scheduling and governing on mobile devices with big.LITTLE processors," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 1, article 17, pp. 1–23, 2016.

[12] Y. Huang, Z. Zha, M. Chen, and L. Zhang, "Moby: A mobile benchmark suite for architectural simulators," in *Proc. Int. Symp. Performance Anal. Syst. Softw.*, 2014, pp. 45–54.

[13] T. Hruby, H. Bos, and A. S. Tanenbaum, "When slower is faster: On heterogeneous multicores for reliable systems," in *Proc. USENIX Annu. Techn. Conf.*, 2013, pp. 255–266.

[14] D. Kadjo, U. Y. Ogras, R. Z. Ayoub, M. Kishinevsky, and P. Gratz, "Towards platform level power mnagement in mobile systems," in *Proc. Int. SoC Conf.*, 2014, pp. 146–151.

[15] M. U. K. Khan, M. Shafique, and J. Henkel, "Hierarchical power budgeting for dark silicon chips," in *Proc. Int. Symp. Low Power Electron. Des.*, 2015, pp. 213–218.

[16] H. Khdr, S. Pgani, M. Shafique, and J. Henkel, "Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips," in *Proc. 52nd Des. Autom. Conf.*, 2015, pp. 1–6, Art. no. 179.

[17] H. Khdr, et al., "Power density-aware resource management for heterogeneous tiled multicores," *IEEE Trans. Comput.*, vol. 66, no. 3, pp. 488–501, Mar. 2017.

[18] J. M. Kim, S. K. Seo, and S. W. Chung, "Looking into heterogeneity: When simple is faster," in *Proc. 2nd Int. Workshop Parallelism Mobile Platforms*, 2014, pp. 1–2.

[19] M. Kim, Y. G. Kim, and S. W. Chung, "Measuring variance between smartphone energy consumption and battery life," *IEEE Comput.*, vol. 47, no. 7, pp. 59–65, Jul. 2014.

[20] Y. G. Kim, M. Kim, J. M. Kim, and S. W. Chung, "M-DTM: Migration-based dynamic thermal management technique for heterogeneous mobile multi-core processors," in *Proc. Des. Autom. Test Europe Conf.*, 2015, pp. 1533–1538.

[21] W.-Y. Liang and P.-T. Lai, "Design and implementation of a critical speed-based DVFS mechanism for the android operating system," in *Proc. Int. Conf. Embedded Multimedia Comput.*, 2010, pp. 1–6.

[22] C.-H. Lin, P.-C. Hsiu, and C.-K. Hsieh, "Dynamic backlight scaling optimization: A cloud-based energy-saving service for mobile streaming applications," *IEEE Trans. Comput.*, vol. 63, no. 2, pp. 335–348, Feb. 2014.

[23] G. Liu, J. Park, and D. Marculescu, "Procrustes$^1$: Power constrained performance improvement using extended maximize-then-swap algorithm," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 34, no. 10, pp. 1664–1676, Oct. 2015.

[24] S. Mittal, "A survey of techniques for architecting and managing asymmetric multicore processors," *ACM Comput. Surveys*, vol. 48, no. 3, article 45, pp. 1–38, 2016.

[25] N. C. Nachiappan, et al., "Domain knowledge based energy management in handhelds," in *Proc. Int. Symp. High Performance Comput. Archit.*, 2015, pp. 150–160.

[26] V. Pallipadi and A. Starikovskiy, "The ondemand governor," in *Proc. Linux Symp.*, 2006, pp. 215–230.

[27] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra, "Power-performance modeling of mobile gaming workloads on heterogeneous MPSoCs," in *Proc. 52nd Annu. Des. Autom. Conf.*, 2015, Art. no. 201.

[28] A. Pathania, S. Pagani, M. Shafique, and J. Henkel, "Power management for mobile games on asymmetric multi-cores," in *Proc. Int. Symp. Low Power Electron. Des.*, 2015, pp. 243–248.

[29] V. Petrucci, O. Loques, D. Mosse, R. Melhem, N. A. Gazala, and S. Gobriel, "Energy-efficient thread assignment optimization for heterogeneous multicore systems," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 1, article 15, pp. 1–26, 2015.

[30] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, "Improving mobile gaming performance through cooperative CPU-GPU thermal management," in *Proc. 53rd Des. Autom. Conf.*, 2016, Art. no. 47.

[31] W. Quan and A. D. Pimentel, "A Sceniario-based run-time task mapping algorithm for MPSoCs," in *Proc. 50th Annu. Des. Autom. Conf.*, 2013, Art. no. 131.

[32] S. Ren, Y. He, S. Elnikety, and K. McKinley, "Exploiting processor heterogeneity for interacting services," in *Proc. Int. Conf. Autom. Comput.*, 2013, pp. 45–58.

[33] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The EDA challenges in the dark silicon Era," in *Proc. 51st Des. Autom. Conf.*, 2015, pp. 1–6.

[34] M. Shafique, D. Gnad, S. Garg, and J. Henkel, "Variability-aware dark silicon management in on-chip many-core systems," in *Proc. Des. Autom. Test Europe Conf.*, 2015, pp. 387–392.

[35] D. Shelepov, et al., "HASS: A scheduler for heterogeneous multi-core systems," *ACM SIGOPS Operating Syst. Rev.*, vol. 43, pp. 66–75, 2009.

[36] D. Stamoulis and D. Marculescu, "Can we guarantee performance requirements under workload and process variations?" in *Proc. Int. Symp. Low Power Electron. Des.*, 2016, pp. 308–313.

[37] P.-H. Tseng, P.-C. Hsiu, C.-C. Pan, and T.-W. Kuo, "User-centric energy-efficient scheduling on multi-core mobile devices," in *Proc. 51st Annu. Des. Autom. Conf.*, 2014, pp. 1–6.

[38] S. Zhuravlev, J. C. Saez, S. Blagodurov, A. Fedorova, and M. Prieto, "Survey of scheduling techniques addressing shared in multicore processors," *ACM Comput. Surveys*, vol. 45, no. 1, article 4, pp. 1–28, 2012.

[39] Emarketer, 2 Billion Consumers Worldwide to Get Smartphones by, 2016. [Online]. Available: http://www.emarketer.com/Article/2-Billion-Consumers-Worldwide-Smartphones-by-2016/1011694

[40] Ericsson, Ericsson Mobility Report Q1, (2015). [Online]. Available: http://www.ericsson.com/res/docs/2015/ericsson-mobility-report-june-2015.pdf

[41] Hardkernel, Odroid-XU3. (2014). [Online]. Available: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127

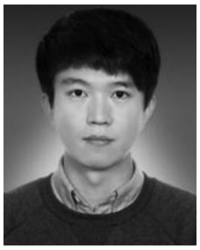
[42] Linux, CPUFreq Governors. (2012). [Online]. Available: https://android.googlesource.com/kernel/comon/+/a7827a2a60218b25f222b54f77ed38f57aebe08b/Documentation/cpu-freq/governors.txt

[43] Linux, Energy-Aware Scheduling (EAS) Project. (2015). [Online]. Available: http://www.linaro.org/blog/core-dump/energy-aware-scheduling-eas-project/

[44] Linux, GTS - A Solution to Support ARM's big.LITTLE Technology. (2014). [Online]. Available: http://www.slideshare.net/linaroorg/lca14-104-gtsasolutiontoarmsbiglittletechnology

[45] Monsoon Solutions, Monsoon Power Monitor. (2017). [Online]. Available: http://www.msoon.com/LabEquipment/PowerMonitor/

[46] Samsung Electronics, Exynos 5422. (2014). [Online]. Available: http://www.samsung.com/global/business/semiconductor/product/application/detail?productId=7978&iaId=234



**Young Geun Kim** received the BS degree from the Division of Computer and Communication Engineering, Korea University, in 2014. He is currently working toward the PhD degree in the Department of Computer Science, Korea University. His research interests include power/thermal management for mobile devices.

**Minyong Kim** received the BS, MS, and PhD degrees in computer science from Korea University, in 2010, 2012, and 2016 respectively. He is currently working toward the MS degree in the Graduate School of Design, Harvard University. His research interests include low-power design, user-aware design, and system-on-chip design. He is a student member of the IEEE.

**Sung Woo Chung** received the BS, MS, and PhD degrees in electrical engineering and computer science from Seoul National University, in 1996, 1998, and 2003, respectively. He is currently a professor in the Department of Computer Science, Korea University. His research interests include low-power design, temperature-aware design, and user-aware design. He was an associate editor of the *IEEE Transactions on Computers* from 2010 to 2015. He was the Technical Program co-chair of the IEEE International Conference on Computer Design in 2015. He serves (and served) on the technical program committees in many conferences, including Design Automation Conference (2015-2017) and International Symposium on Low Power Electronics and Design (2016-2017). He is a senior member of the IEEE.