







Power-Efficient Big.LITTLE Core Assignment Scheme for Task Graph Based Real-Time Smartphone Applications

Se Won Lee¹ , Donghoon Kim² , and Sung-Hwa Lim³  

¹ Pukyong National University, Busan, Republic of Korea
swlee@pknu.ac.kr

² TmaxSoft, Seongnam, Republic of Korea
ldh94@ajou.ac.kr

³ Namseoul University, Cheonan, Republic of Korea
sunghwa@nsu.ac.kr

Abstract. Demand of energy saving for smartphone batteries is increasing along with the improvement in quality and performance of smartphone applications. In response to these demand, most of the smartphones recently released are equipped with ARM big.LITTLE architecture, which is composed of relatively energy efficient low performance cores (LITTLE cores) and high power consumption high performance processor cores (big cores). However, it is difficult to take full advantage of the energy-saving benefits of the ARM big.LITTLE architecture, because most real-time tasks tend to be assigned to big cores rather than LITTLE cores. To solve this problem, we propose power-efficient multi-core allocation schemes for task graph-based real-time smartphone applications that can increase the utilization of LITTLE cores. The experiment on an off-the-shelf smartphone have shown that the algorithm can reduce energy consumption by up to 50% while meeting the applications deadline. We also discuss energy-aware security issues on big.LITTLE core assignments of real-time application threads.

Keywords: Energy conservation · Asymmetric multi-cores · Smartphone · Real-time applications

1 Introduction

With the spread of smartphones and the emergence of high-quality applications, there is a growing interest in the issues of power consumption. Recently, as smartphones incorporate IoT, Augmented Reality and Virtual Reality (AR/VR), the more demand for processing high workloads on smartphones in real time [2, 5]. Moreover, a smartphone play an important role in personal area networks as a network edge or a sensor edge. A Real-time system typically has a deadline

for each task, ensuring that it is processed within each deadline. To meet this time constraint (i.e., deadline), the system may require higher performance, which consumes more power. However, advances in battery technology tend to lag behind the development levels of such high power hardware and applications [15].

To address this problem, the asymmetric multi-core architecture is being introduced into the mobile environment, which consists of multiple processors having different processing power and different power efficiency. A widely used asymmetric multi-core architecture is the big.LITTLE architecture developed by ARM [8]. The ARM big.LITTLE architecture combines big cores with high processing power and high power consumption, and LITTLE cores with low power consumption and relatively low performance. We can increase the energy efficiency by allocating the application tasks that are not urgent and do not require high throughput to LITTLE cores. However, the desired energy savings by the big.LITTLE architecture will not be sufficiently achieved unless the LITTLE cores are fully utilized. In practice, many application tasks have performed on the big cores instead of the LITTLE cores [17]. This is because the criteria for assigning an application's tasks to cores are based on the task's priority and the load. Unfortunately, most applications have high priorities to meet user requirements.

Tasks with deadlines in real-time applications are considered urgent so that they tend to run on big cores. However, real-time tasks do not affect performance or satisfaction even though the processing time increases as long as the deadline is guaranteed. Therefore, in order to increase the energy saving effect of the big.LITTLE core structure, a novel energy efficient multi-core assignment technique is required to increase the utilization of LITTLE core in application environment with real-time characteristics. A running application consists of a group of task, which has an acyclic graph structure. We propose a power-efficient big.LITTLE core assignment technique that estimates the deadline compliance status of task graph based real-time application, and assigns the guaranteed tasks to LITTLE core first. We also consider guarantee of the deadline of a task graph for an application. By applying the proposed scheduling technique on the real test-bed, we show that the proposed technique improves energy saving effect while guaranteeing real-time performance as compared to the performance of the legacy scheduler.

Since the battery power is one of the crucial resources of smartphones, to make a smartphone's battery power quickly discharged may be one of the effective way for malicious attackers [12]. Our work can provide a thread-level energy consumption effect on an application task. Therefore, it may be utilized to detect or diagnose these kinds of battery power attacks. We also discuss energy-aware security issues on big.LITTLE core assignments of real-time application threads.

The paper is organized as follows. In Sect. 2, we discuss related studies that have been used to reduce energy consumption in asymmetric multi-core architectures. Section 3 and 4 describe the system model and the multi-core assignment algorithm proposed in this paper, respectively. Section 5 shows the

performance evaluation with experimental results of the proposed scheme. We discuss energy security issues in Section 6, and conclude the paper in Sect. 7.

2 Related Work

Since ARM Holdings introduced ARM big.LITTLE structure, many studies have been continued on ARM big.LITTLE structure. In [8], the big.LITTLE architecture’s power efficiency is introduced by comparing the legacy symmetric multi-core architecture. The software structure for task scheduling with these asymmetric multi-core equipped device can be divided into three main categories [7,8]. Figure 1 depicts each technique.

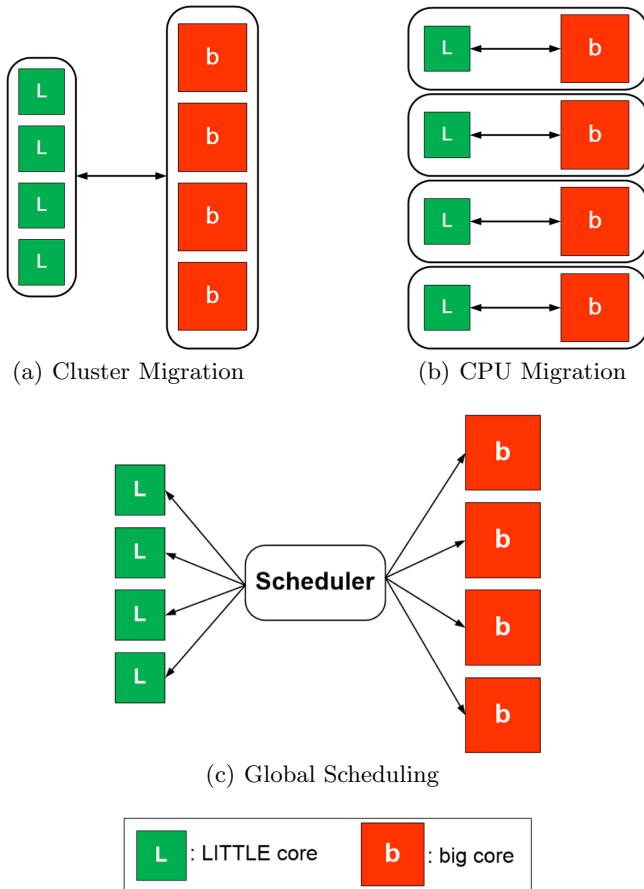


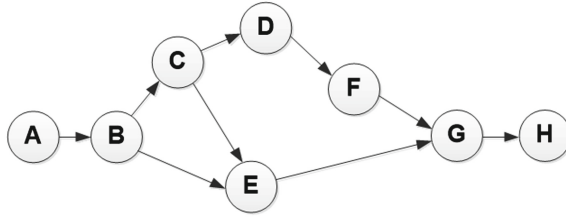
Fig. 1. big.LITTLE software models [4,8]

- Cluster Migration Technique: Multiple big cores make up a cluster, and several LITTLE cores make up another cluster. When the scheduler runs the task, the scheduler selects either the big core cluster or the LITTLE core cluster. The deselected cluster becomes inactive. The disadvantage is that not all cores are available at the same time.
- CPU Migration Technique: In an environment where there are multiple big and LITTLE cores, one big core and one LITTLE core are paired together to form a cluster. In situations where there are many tasks to run, all clusters can be used simultaneously, but within one cluster only one of the big and LITTLE cores is active, and the other cores are inactive.
- Global Task Scheduling Technique: Each core is independent, and the scheduler assigns tasks to the appropriate core (big core or LITTLE core) according to the schedule policy setting. If the scheduler has a lot of tasks to run, the scheduler can activate all cores. Since the scheduling complexity is higher than that of existing techniques, careful scheduling techniques are required.

The studies in [14,17] present detailed comparisons between the ARM big.LITTLE structure and the legacy high performance CPUs for performance and energy trade-offs. In the study [1,18], the authors proposed application assisted core assignment technique for the ARM big.LITTLE structure to save energy especially running the web browser. The study in [6] proposed an energy efficient heuristic scheme for the big.LITTLE core architecture to schedule multiple applications using offloading. However, since the study [6] focuses on scheduling through offloading, it does not suggest a method to increase the utilization of LITTLE cores. Our research group presented an energy efficient multi-core assignment algorithm exploiting LITTLE cores as long as every task in a task graph (i.e., application) can complete its execution before the deadline of the task is met [10]. However, the guarantee of the deadline of a task graph is not considered in [10].

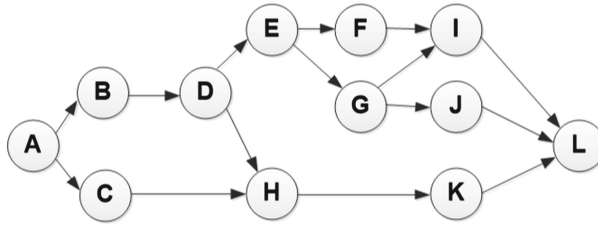
3 System Model

In our scheduling model, each application m has multiple tasks, and we use a directed acyclic graph to represent an application with its tasks. In the graph, each node is represented $X_{m,i}$ for application m , and task i . In addition, the connection between each node represents dependency between the nodes. For example, if there is a dependency between $X_{m,i}$ and $X_{m,j}$ ($X_{m,j}$ follow $X_{m,i}$), $X_{m,j}$ can only be executed after the completion of $X_{m,i}$. Figure 2 shows the task graph for two applications. The two applications are independent of each other, and each node, or task, has dependency in each application. Also it presents the job size and the deadline for each task. For example, task D of Application 1 in Fig. 2 can start its execution only after all precedent tasks of task D (i.e., task A, B, and C) are completed. The execution of an application is presented as the time spend in completing the first node through the last node. For example, the execution time of Application 2 in Fig. 2 is the time spend in completing task A through task L.



Task	Expected Execution Time (s.)		Deadline (s.)
	big	LITTLE	
A	3	6	8
B	3	6	8
C	2	4	4
D	3	6	8
E	5	10	12
F	8	16	16
G	5	10	12
H	4	8	8

(a) Application 1



Task	Expected Execution Time (s.)		Deadline (s.)
	big	LITTLE	
A	3	6	8
B	2	4	4
C	5	10	12
D	7	14	16
E	5	10	12
F	6	12	12
G	2	4	4
H	7	14	12
I	5	10	12
J	3	6	8
K	2	4	4
L	2	4	5

(b) Application 2

Fig. 2. Examples of task graph

4 Power Efficient Multi-core Assignment Algorithms

4.1 Meeting the Deadline of Each Task

The basic idea is that if a task can finish its execution running on a little core before its deadline, the task is assigned on the little core. The algorithm progresses as shown in Algorithm 1. A task graph for an application to be scheduled is required as input data. The main procedure and the essential functions are described as follows:

- Main procedure: A task is taken from the input task graph in descending order. It checks if the task is ready, i.e., all of precedent tasks of the task should be completed. If the task is ready, the task is assigned to a LITTLE core only if the task can be completed within the deadline. Otherwise, the task is assigned to a big core.
- *ExpectedExecTime(task, coreType)*: This function computes the expected execution time to complete the given *task* while running on the *coreType*. *coreType* can be either big or LITTLE.
- *getAvailableCoreList (coreType)*: This function returns the list of available cores among *coreType* (i.e., big or LITTLE).

Algorithm 1: Multi-core assignment algorithm considering the deadline of each task

```

1 Input: Task Graph  $G$ 
2 while  $G$  is not empty do
3    $T_i \leftarrow$  next task from  $G$ 
4   while precedent task of  $T_i$  is not completed do
5      $T_i \leftarrow$  next task from  $G$ 
6   if  $ExpectedExecTime(T_i, LITTLE) < deadline$  of  $T_i$  then
7      $S_{available} = getAvailableCoreList(LITTLE)$ 
8   else
9      $S_{available} = getAvailableCoreList(big)$ 
10  if  $S_{available}$  is not empty then
11    Assign  $T_i$  to a core of  $S_{available}$ 
12  else
13    Assign  $T_i$  to any available core
14  remove  $T_i$  from  $G$ 

```

4.2 Meeting the Deadline of the Application

Algorithm 1 presented in the previous section guarantees that each task completes its execution within its deadline as shown. For example, it is guaranteed that *Execution time of a task* shown in Fig. 3 should always be smaller than the

required deadline of the task. However, it is more usual that an application (i.e., its task graph) has to be finished within its deadline. For example, *Execution time of an Application* in Fig. 3 should always be smaller than the required deadline of the task graph (i.e., the application).

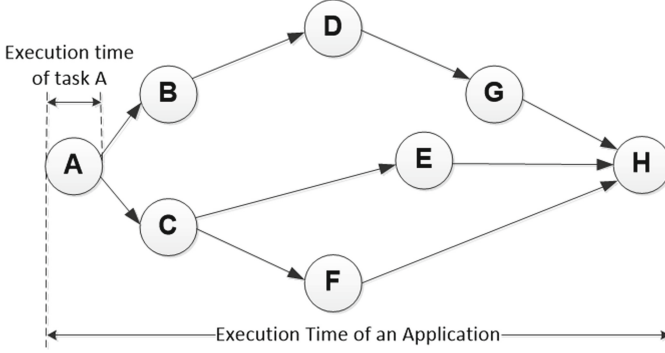


Fig. 3. Execution time of a task and an application

In Algorithm 2, we present a multi-core assignment algorithm considering the deadline of a task graph.

Algorithm 2: Multi-core assignment algorithm considering the deadline of a task graph

```

1 Input: Task Graph  $G$ 
2  $CriticalPath_G \leftarrow FindCriticalPath(G)$ 
3 while  $G$  is not empty do
4    $T_i \leftarrow$  next task from  $G$ 
5   while precedent task of  $T_i$  is not completed do
6      $T_i \leftarrow$  next task from  $G$ 
7   if  $T_i$  is a vertex of  $CriticalPath_G$  then
8      $ExecTimeSum_c \leftarrow$  sum of all vertex's expected execution time of
       $CriticalPath_G$ 
9     if  $ExecTimeSum_c$  is greater than the deadline of  $G$  then
10       $coreType \leftarrow big$ 
11     else
12       $coreType \leftarrow LITTLE$ 
13      $S_a\ available = getAvailableCoreList(coreType)$ 
14     if  $S_a\ available$  is not empty then
15       Assign  $T_i$  to a core of  $S_a\ available$ 
16     else
17       Assign  $T_i$  to any available core
18     remove  $T_i$  from  $G$ 

```

The main procedure and the essential function in Algorithm 2 are described as follows:

- Main procedure: At first, it finds out the critical path, which is explained in the next paragraph, from the given graph. In the loop, a task is taken from the input task graph in descending order. It checks if the task is in the critical path, the task is assigned to a big core. Otherwise, the task is assigned to a LITTLE core only if the task can be completed within the deadline. If it can not be, the task is assigned to a big core.
- *FindCriticalPath*(Graph G): This function finds the critical path of the given graph G . A critical path of a graph includes a path from the first node (i.e., root node) to the last node (i.e., terminal node), of which the sum of execution time of the member nodes is greater than any other path of the graph.

5 Performance Evaluation

In this chapter, we evaluate the performance of our proposed scheme by implementing a test program on a real smartphone. In the evaluation, we randomly generate task graphs, then proceed with scheduling according to the proposed algorithm. Using this randomly generated task graph, we compared our scheduler with the Android’s default scheduler. Before showing the main results, we first present the experiment setting.



Fig. 4. Experimental equipment

5.1 Experiment Setup

We conducted experimental measurements by implementing our proposed scheme on Samsung Galaxy S7 edge [16], an off-the-shelf smartphone. Samsung Galaxy S7 edge embeds four big cores and four LITTLE cores. As a default setting, the clock speed of the LITTLE core is set at 1.6 GHz and that of the big core is 2.3 GHz. We measure the energy consumption of the smartphone during the experiment by using Monsoon HV power monitor [13], as shown in Fig. 4. For the clarification for measurements, we removed the smartphone’s battery and powered directly from the monsoon power monitor. The smartphone was set in the airplane mode during the experiment to prevent other functions working.

In the experiment, we assume that the application can have 8, 16, 20 and 24 tasks, each of which we measure the energy consumption until the end of the application. Each task runs the Linpack algorithm [4]. The task graph of the application program was randomly generated using TGFF [3] for each task number (i.e., 8, 16, 20, 24). For the same environment of experiments, we used a group of randomly generated task graphs with similar CPU utilization.

5.2 Experiment Result

The experiment increases the number of tasks to 8, 16, 20, 24 under the fixed CPU *Utilization* (ρ) as shown in Fig. 5, 6, and 7. And then we measure the amount of change in the power consumption of the existing scheduling technique (i.e., *legacy*) and our proposed scheduling technique (i.e., *Power-Efficient*).

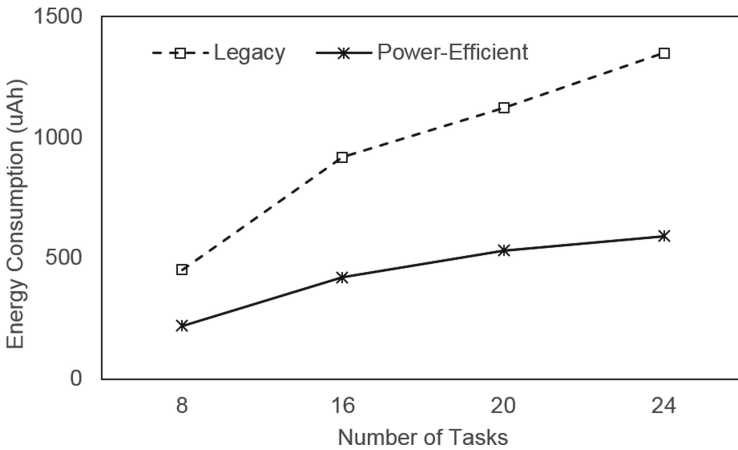


Fig. 5. Energy consumption for varying number of tasks when $\rho = 25\%$

Figure 5 illustrates the result when the amount of tasks to be run is small (i.e., average CPU *Utilization* is 25%). Because *Utilization* is 25%, most of the tasks are assigned at the LITTLE core in *Power-Efficient*, whereas in *legacy*,

most of the tasks are assigned at the big core. The more number of tasks, the more power saving effect will be in *Power-Efficient* than *legacy*. It is found that *Power-Efficient* reduces energy consumption by 50% compared to *legacy* when the number of tasks is 24.

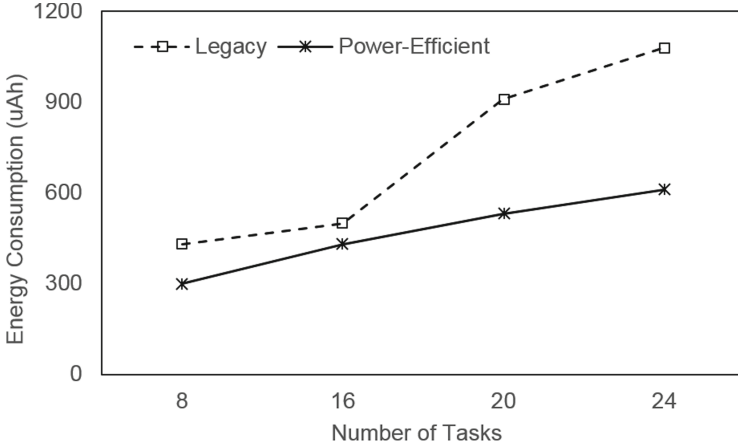


Fig. 6. Energy consumption for varying number of tasks when average $\rho = 50\%$

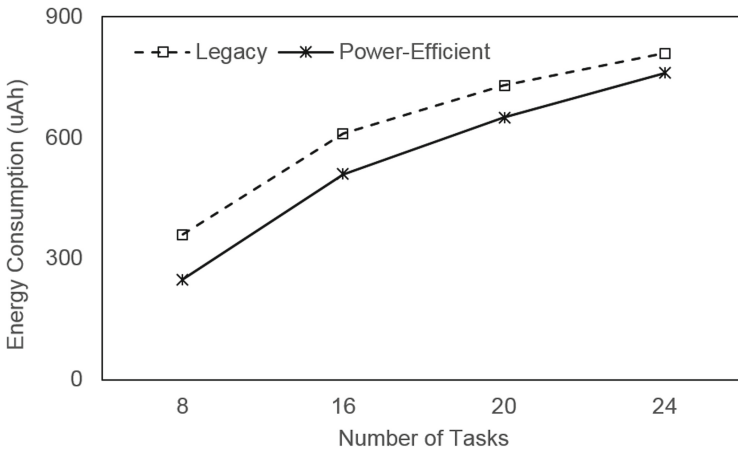


Fig. 7. Energy consumption for varying number of tasks when average $\rho = 75\%$

Figure 6 shows the result when the amount of tasks to be run is medium (i.e., average CPU Utilization is 50%). Because more tasks can be handled at the same time than the case of Fig. 5, energy consumption in both *legacy* and

Power-Efficient decreases than the result of Fig. 5. Similarly, as the number of tasks increases, the difference of energy consumption between *Power-Efficient* and *legacy* increases. When 24 tasks are used, we can find that *Power-Efficient* reduces 42% energy consumption compared to *legacy*.

Figure 7 illustrates the result when CPU *Utilization* is 75%. Since *Utilization* is increased compared to the case of Fig. 6, the energy consumption of both schemes are reduced. Though the difference of energy consumption between the two scheduling techniques decreases as the number of tasks increases, the *Power-Efficient* is at least 5% less energy consumption than *legacy*.

6 Discussion of Energy-Aware Security Issues

Since the battery power is one of the crucial resources of smartphones, making a smartphone wastefully consume its battery power to be quickly discharged may be one of the most effective ways for malicious attackers. Each applications of a smartphone consists of several real-time threads some of which have dependency of the execution (i.e., thread dependency graph of a task). Therefore, if a smartphone employing the application assisted big.LITTLE core assignment policy is hacked by an attacker, its battery can be wasted by assigning one of application threads on wrong cores. If a thread which can be run on a LITTLE core is forced to be assigned on a big core, the energy consumption will be increased. On the other hand, if a thread which should be run in real-time is forced to be assigned on a LITTLE core, deadline misses will be occurred. The application may start over the thread if its deadline is missed, which may also increase the power consumption. Therefore, in order to maintain strict security against malicious attacks, it is desirable to use power efficient and time-aware big.LITTLE core assignments.

7 Conclusion and Future Works

In this paper, we propose power efficient multi-core assignment schemes that process real-time tasks in asymmetric multi-core mobile devices while guaranteeing not only the deadlines of each tasks but also the deadline of the given task graph. To evaluate the performance (i.e., energy saving effect) of the proposed multi-core assignment scheme, the proposed algorithms are implemented and measured on an off-the-shelf smartphone. The experimental results show that the proposed scheme reduces the energy consumption by up to 50% compared to the conventional scheduling technique when the CPU *Utilization* is less than 25%, and also show that the proposed scheme reduces the energy consumption by at least 5% even if both the *Utilization* and the number of tasks increases.

The proposed scheme is heavily influenced by the prediction accuracy of the expected processing time of the task before a task is assigned to one of multi-cores. Therefore, as a future work, we will employ a machine learning techniques such as support vector machine (SVM) to enhance the prediction accuracy of the expected processing time of a task in the proposed scheme [9, 11].

References

1. Bui, D.H., Liu, Y., Kim, H., Shin, I., Zhao, F.: Rethinking energy-performance trade-off in mobile web page loading. In: Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, pp. 14–26 (2015)
2. Chow, Y.W., Susilo, W., Phillips, J.G., Baek, J., Vlahu-Gjorgievska, E.: Video games and virtual reality as persuasive technologies for health care: an overview. *JoWUA* **8**(3), 18–35 (2017)
3. Dick, R.P., Rhodes, D.L., Wolf, W.: TGFF: task graphs for free. In: Proceedings of the 6th International Workshop on Hardware/Software Codesign, CODES/CASHE 1998, pp. 97–101. IEEE (1998)
4. Dongarra, J.J., Luszczek, P., Petitet, A.: The LINPACK benchmark: past, present and future. *Concurr. Comput. Pract. Exp.* **15**(9), 803–820 (2003)
5. Fiorino, D., Collotta, M., Ferrero, R.: Usability evaluation of touch gestures for mobile augmented reality applications. *J. Wirel. Mob. Netw. Ubiqui. Comput. Dependable Appl. (JoWUA)* **10**(2), 22–36 (2019)
6. Geng, Y., Yang, Y., Cao, G.: Energy-efficient computation offloading for multicore-based mobile devices. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 46–54. IEEE (2018)
7. Ghasemi, H.R., Karpuzcu, U.R., Kim, N.S.: Comparison of single-ISA heterogeneous versus wide dynamic range processors for mobile applications. In: 2015 33rd IEEE International Conference on Computer Design (ICCD), pp. 304–310. IEEE (2015)
8. Greenhalgh, P.: big.LITTLE technology: The future of mobile. ARM Limited, White Paper, p. 12 (2013)
9. Hsu, C.W., Chang, C.C., Lin, C.J., et al.: A practical guide to support vector classification (2003)
10. Kim, D.H., Ko, Y.B., Lim, S.H.: Energy-efficient real-time multi-core assignment scheme for asymmetric multi-core mobile devices. *IEEE Access* **8**(1), 117324–117334 (2020)
11. Kotenko, I.V., Saenko, I., Branitskiy, A.: Applying big data processing and machine learning methods for mobile internet of things security monitoring. *J. Internet Serv. Inf. Secur.* **8**(3), 54–63 (2018)
12. Merlo, A., Migliardi, M., Caviglione, L.: A survey on energy-aware security mechanisms. *Pervasive Mob. Comput.* **24**, 77–90 (2015)
13. Monsoon Solutions Inc.: High voltage power monitor (March 2019). <https://www.monsoon.com>
14. Padoin, E.L., Pilla, L.L., Castro, M., Boito, F.Z., Navaux, P.O.A., Méhaut, J.F.: Performance/energy trade-off in scientific computing: the case of arm big.LITTLE and Intel Sandy Bridge. *IET Comput. Digit. Tech.* **9**(1), 27–35 (2014)
15. Paradiso, J.A., Starner, T.: Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Comput.* **4**(1), 18–27 (2005)
16. Park, J., et al.: Mobile phone, US Patent App. 29/577,834, 11 April 2017
17. Seo, W., Im, D., Choi, J., Huh, J.: Big or little: a study of mobile interactive applications on an asymmetric multi-core platform. In: 2015 IEEE International Symposium on Workload Characterization, pp. 1–11. IEEE (2015)
18. Zhu, Y., Reddi, V.J.: High-performance and energy-efficient mobile web browsing on big/little systems. In: 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), pp. 13–24. IEEE (2013)