User Interaction Aware Reinforcement Learning for Power and Thermal Efficiency of CPU-GPU Mobile MPSoCs

Somdip Dey[1], Amit Kumar Singh[1], Xiaohang Wang[2] and Klaus McDonald-Maier[1]
[1]Embedded and Intelligent Systems Laboratory, *University of Essex*
[2]School of Software Engineering, *South China University of Technology*
Email: { [1] somdip.dey, [1] a.k.singh, [1] kdm }@essex.ac.uk; [2] baikeina@163.com

***Abstract*—Mobile user's usage behaviour changes throughout the day and the desirable Quality of Service (QoS) could thus change for each session. In this paper, we propose a QoS aware agent to monitor mobile user's usage behaviour to find the target frame rate, which satisfies the desired user's QoS, and applies reinforcement learning based DVFS on a CPU-GPU MPSoC to satisfy the frame rate requirement. Experimental study on a real Exynos hardware platform shows that our proposed agent is able to achieve a maximum of 50% power saving and 29% reduction in peak temperature compared to stock Android's power saving scheme. It also outperforms the existing state-of-the-art power and thermal management scheme by 41% and 19%, respectively.**

***Index Terms*—agent system, reinforcement learning, machine learning, CPU, GPU, power optimization, thermal optimization, MPSoCs, user behaviour, user interaction, smartphone, mobile**

I. Introduction

Due to fast advancement in chip integration technology in the past couple of decades, we can see an increased adaptation of heterogeneous multi-processor System-on-Chip (MPSoC) in Edge devices, especially in smartphones and tablets. Market research performed by eMarketer [1] shows that in 2018 mobile users in the USA spend 4 hours 16 minutes on an average on in-apps and mobile web with the availability of the mobile Internet on their smartphones and tablets. This includes an average of 1 hour 56 minutes on the top 5 social media platforms: Youtube, Facebook, Snapchat, Instagram and Twitter [2]. Another market research by Deloitte US and Rescue Time [3], [4] shows that an average person picks-up/look at their phones 52 times during their workday, where 70% of the sessions are less than 2 minutes, 25% of the sessions lasting between 2 to 10 minutes and 5% of the sessions prolonged more than 10 minutes. Even the duration of the user picking-up/looking at their Edge device every time varies from user to user and hence, making the sessions stochastic in nature and furthermore making existing resource management (DPTM) techniques inefficient for real-world Quality of Service driven applications on Edge devices.

Fig. 1 shows the varying frames per second (FPS) generation (frame rate denoted as *schedutil FPS* in the primary vertical axis) while using home screen, facebook and spotify apps on Samsung Galaxy Note 9 [5], which employs Exynos 9810 MPSoC [6], over a 5 minutes of session. *Frame rate* is the frequency at which a new frame is rendered. In Fig. 1, FPS is recorded and shown every 3 seconds to provide a holistic view on the variation of frame rate during the session, especially the frame rate could vary even for one application based on the user's usage behaviour. In Fig. 1, the secondary vertical axis shows the operating frequency of the big and LITTLE CPUs of Exynos 9810 MPSoC, where the *Freq. B. sched* denotes the operating frequency of the big cores, and *Freq. L. sched* denotes the operating frequency of the LITTLE cores.

This work is supported by the UK Engineering and Physical Sciences Research Council EPSRC [EP/R02572X/1 and EP/P017487/1], Natural Science Foundation of Guangdong Province No. 2018A030313166, Pearl River S&T Nova Program of Guangzhou No. 201806010038, the Fundamental Research Funds for the Central Universities No. 2019MS087, and the Natural Science Foundation of China No. 61971200.

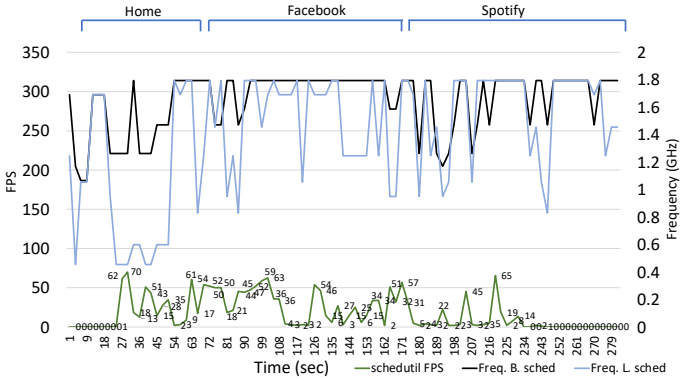

Fig. 1. FPS generation, operating frequency of big and LITTLE CPUs in Samsung Note 9 while using home screen, Facebook and Spotify apps during a session on schedutil governor

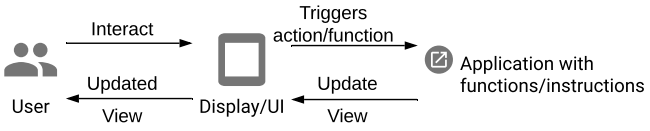

Fig. 2. Interaction between the user and an app on a smartphone happens through display/UI

User experience or Quality of Service (QoS) of an application on an Edge device, especially smartphones, is often a measure of their frame rates. On modern smartphones, the user interacts with the display/UI which then triggers an event (action), which is again tied to some function(s) or instructions of the application with which the user is interacting [7], [8], as shown in Fig. 2. Contrary to popular belief that the QoS of games or media applications, such as videos, are the only applications where the user experience is evaluated through their frame rates, the user's experience of any application on a modern smartphone with a touch screen display is measured with the frame rate as well. As the frame rate increases, the display experience tends to appear smoother and more fluid to the human eye and hence, very high frames-per-second (FPS) is often translated to a much richer experience for the user. Typically most commercial mobile devices render a maximum of 60 FPS to match their display's refresh rate of 60 Hz. A display *refresh rate* is the frequency at which the display is updated. Although at the moment there are some commercial devices which have higher display refresh rate such as 90 Hz, 120 Hz, 60 Hz display refresh rate continues to be the most popular mobile display refresh rate available in the majority of the devices.

The refresh rate and the frame rate are synchronized to update the display of the device through the process of Vertical Synchronization (VSync) [7]. On the Android OS, 3 buffers consisting of 1 front buffer and 2 back buffers are used for VSync. CPU/GPU renders the new frames in the back buffers while the display shows the content of the front buffer. When a new frame is rendered in the back buffer, after each VSync the content of the back buffer is pushed to the front buffer and hence the display outputs the front buffer content to the user. Since most displays have a 60 Hz refresh rate, the VSync is generated every 16.67 ms for such devices. The display is only refreshed on each VSync regardless of whether new frames are generated within the VSync period. When CPU/GPU fails to produce a frame within the VSync period, the front buffer is not updated and the display continues to render the previous frame, which results in a drop of the frame (*frame drop*) and hence, hindering the user experience. These frame drops lead to lag or stutter and hence reduced QoS is achieved. Every mobile application on smartphones is a dynamic application consisting of periodic, aperiodic and sporadic tasks [9], where the load of the application constantly varies based on the user interaction and the mechanics of the application itself.

For example, the primary vertical axis of Fig. 1 shows that for the same (intra-) mobile application (Facebook or Spotify) different FPS is generated at the different time period during the session based on varied interaction between the user and the application through the display/UI. If we take a closer look at the operating frequency of the big and LITTLE CPU cores (as shown in the secondary vertical axis of Fig. 1) of the Exynos 9810 MPSoC while using the applications (Facebook or Spotify) we could notice that the operating frequency remains relatively very high yet generating less FPS at certain occasions (as is evident in Fig. 1). This phenomenon is more evident while using the Spotify app during the session where the FPS drops close to 0 yet the operating frequencies of the big and LITTLE cores are very high, which results in high power consumption and operating temperature of the device.

Several power and thermal management schemes [10]–[20] for power and thermal efficiency while considering frame rate or QoS have been proposed over the years. However, neither of the techniques tries to improve power and thermal efficiency by taking user's interaction with the mobile into account to cater for improved QoS. Moreover, most of the existing studies focus on maximizing performance per watt (PPW), however, for a mobile platform reducing power consumption as well as the temperature of the device is very important while catering for the performance requirement and trying to maximize PPW is not enough for such platform. To overcome this limitation in this paper we also introduce a new metric to incorporate performance, power consumption and thermal behaviour of the mobile device.

A. *Efficacy of our proposed technique*

In this paper, we propose a reinforcement learning based intelligent agent, called *Next* (Next generation user interaction aware DVFS), that learns the user's usage pattern of the mobile applications and then utilizes DVFS to save power and reduce the temperature during the mobile usage session while catering for the QoS required by the user. In order to determine the desired QoS for the user for each session, we also define a new metric, which is optimized by Next to improving the reward generated using reinforcement learning. Fig. 3 shows the average power consumption in the primary vertical axis and peak temperature of the big CPUs of Note 9 platform in the secondary vertical axis while utilizing the Next technique for a similar user session using home screen, Facebook and Spotify application. For the similar session using the Next technique we are able to save 41.88% more power (refer to primary vertical axis of Fig. 3) on an average over the time period when compared to default schedutil governor of Android, whereas we are also able to reduce the peak temperature of the big CPU cores by 21.02% (refer to secondary vertical axis of Fig. 3) compared to the temperature of the big cores while on schedutil governor. In mobile MPSoC, the big CPU cores consume the most energy [21] and are also the focus of hot spots on the chip, and hence in this case, we have focused on the thermal behaviour of the big cores for the comparative study.

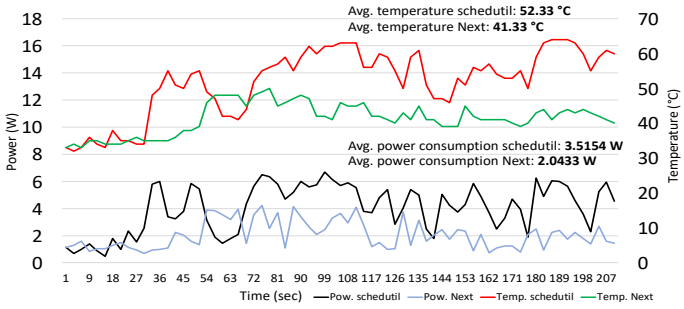

Fig. 3. Power consumption and temperature of big CPUs on Samsung Note 9 while using home screen, Facebook and Spotify apps during a session on schedutil vs Next

B. *Contributions*

The concrete contributions of this paper are the following.

- We define a new metric to optimize QoS based on power consumption and peak temperature obtained.
- To the best of our knowledge, ours is the first work that explores DVFS in mobile CPU and GPU based on user's interaction behaviour using a software agent based on reinforcement learning.
- We implement our power and thermal management technique in the application layer of the Android platform on Galaxy Note 9 smartphone utilizing Exynos 9810 MPSoC, and evaluate its efficacy with the latest popular applications from the Google Play store.

II. State-of-the-art Techniques

Several methodologies [10]–[14] to cater for performance, power consumption and thermal behaviour of mobile devices have been proposed over the years, however, they have their own limitations.

In [10], Pathania et al. proposed an integrated CPU-GPU power management mechanism for 3D mobile games, which requires high FPS to satisfy the QoS of the user. In this methodology, the FPS range, which is assumed to be the required performance in this technique, is averaged over a time period and then CPU and GPU's operating frequencies are set accordingly to cater for its averaged performance while reducing lower power consumption based on a cost model incorporating QoS. This technique has to find out the target FPS as the desired performance in order to evaluate the cost model and set the operating frequency. However, from Fig. 1 we have already noticed that the FPS generation within the same session and for the same application could vary by a large range and hence, utilizing this methodology will lead to much reduced QoS for the users on real commercial devices running multiple applications sequentially.

Sahin et al. [11] proposed a QoS-aware CPU frequency capping mechanism to reduce the peak thermal behvaiour of the mobile devices. In this methodology, the Instructions Per Second (IPS) and the maximum chip temperature is monitored,

and the maximum operating frequency of the CPU is set to keep the temperature within the maximum limit while catering for the desired QoS. However, the mechanism utilizes IPS value of CPU to set the maximum operating frequency of the CPU, which could not be translated to GPU having different utilization capacity based on workload and hence, the methodology is not scalable to GPU nor it has the provision to reduce power consumption reactively or proactively.

An online reinforcement learning based proactive DVFS approach targeting frame-based applications is presented in [12] to improve energy efficiency. In this methodology, the workload is evaluated by the reinforcement learning algorithm and then frequency selection using DVFS is performed as an action to cater for that workload. This methodology could be used for inter- as well intra-application QoS requirement, however, the technique does not consider GPU load towards the QoS requirement, which is an important criteria for gaming applications where the CPU workload could be significantly less yet with very high GPU workload. This work also does not consider reduction in peak temperature of the device.

In [13], Peters et al. proposed a power management strategy for mobile games based on frame- and thread-based workload prediction on MPSoC. This work manages power by using the frame rate and thread workload as metrics to evaluate the appropriate workload predictors and applies thread-to-core mapping along with DVFS to cater for FPS constraint. This work is catered towards gaming applications and not generalized for all types of application. Moreover, this mechanism requires workload prediction, which could be time-consuming in real time for mobile applications and hence induce an overhead, reducing QoS for the users.

Bhat et al. [14] proposed an approach to achieve dynamic power-thermal management in heterogeneous MPSoCs by adapting models for performance, power consumption and temperature of various processing elements in the SoC. This work predicts temperature and power consumption through online learning of GPU frame processing time, GPU power consumption and power-temperature dynamics of a SoC. Several performance counters such as GPU utilization, L2 cache references per instruction, per core CPU utilization are used in this approach, however, none of the performance counters could translate to user's behaviour and usage pattern to provide QoS more suited towards the user's need. Moreover, high GPU or CPU load and utilization does not mean that the frame rate would be high, and vice-versa. For example, when a game/app is loading on the smartphone the FPS drops close to 0 while the splash screen is shown to the user and the CPU/GPU load could be very high in order to reflect all the computation that needs to be done to start the game/app. However, in this scenario running all the CPU-GPU on highest operating frequency could also lead to wasted power consumption. In order to overcome this challenge, we need a mechanism to understand the user's mobile usage along with the user's QoS requirement and the app utilization to set the operating frequencies to cater for the QoS requirement.

III. System, Metric and Problem Formulation

A. Hardware & Software Infrastructure

Nowadays heterogeneous MPSoCs consist of different types of cores, either having the same or different instruction set architecture (ISA). Moreover, the number of cores of each type of ISA can vary based on MPSoCs and are usually clustered if the types of cores are similar. For this research, we have chosen an Asymmetric Multi-processors (AMPs) system-on-chip (AMPSoC), which has clustered cores on the system. We chose to execute our experimental evaluation on Galaxy Note 9 [5], which is the latest mobile device from Samsung and utilizes the Exynos 9810 MPSoC [6]. Exynos 9810 MPSoC has two CPU clusters, one for big CPU cores consisting of 4 Mongoose 3 CPU cores, and the other cluster for LITTLE CPU cores consisting of 4 Cortex A-55 CPU cores. The Mongoose 3 CPU cores allow cluster wise DVFS and has 18 frequency scaling levels ranging from 650 MHz to 2704 MHz (2704 MHz, 2652 MHz, 2496 MHz, 2314 MHz, 2106 MHz, 2002 MHz, 1924 MHz, 1794 MHz, 1690 MHz, 1586 MHz, 1469 MHz, 1261 MHz, 1170 MHz, 1066 MHz, 962 MHz, 858 MHz, 741 MHz, 650 MHz). Similarly, the LITTLE Cortex-A55 CPU cores allow cluster wise DVFS and has 10 frequency scaling levels ranging from 455 MHz to 1794 MHz (1794 MHz, 1690 MHz, 1456 MHz, 1248 MHz, 1053 MHz, 949 MHz, 832 MHz, 715 MHz, 598 MHz, and 455 MHz). Exynos 9810 MPSoC also hosts ARM Mali-G72 MP18 GPU, which has 18 cores operating at a frequency range of 260 MHz to 572 MHz with 6 frequency scaling levels (572 MHz, 546 MHz, 455 MHz, 338 MHz, 299 MHz and 260 MHz). There are 5 thermal sensors on the device, out of which one is placed on the big CPU cluster, and one virtual sensor[1] is used for overall device temperature.

The Galaxy Note 9 was running on Android 9 (Pie) [22] OS utilizing Linux kernel version 4.9.59, which has only one governor named *schedutil* based on Energy Aware Scheduling (EAS) [23].

B. Metric and Problem Definition

The main focus of this work is to meet QoS while optimizing power consumption and peak thermal behaviour based on user's interaction with the application. Most studies available at the moment focus on Performance per watt, however, there is no provision for thermal behaviour in the metric. Therefore, in this work we define a new metric, which incorporates both power consumption and peak temperature to evaluate the performance at a given time period. We call this metric *performance per degree watt* (PPDW), which is represented by the Eq. 1. In Eq. 1, $PPDW_i$ is the performance per degree watt at a time period i, FPS_i, P_i and T_i are the frames-per-second, power consumption and temperature respectively at that time, and T_a is the ambient temperature.

$$PPDW_i = \frac{FPS_i}{\Delta T \times P_i}, \text{where } \Delta T = T_i - T_a \tag{1}$$

The main objective is to minimize the value of PPDW, however, the optimum minimal value, which is $PPDW_{desired}$, needs to be between $PPDW_{worst}$ and $PPDW_{best}$ (refer to Eq. 2), which are defined as: $PPDW_{worst} = \frac{FPS_{least}}{(T_{max}-T_a)\times P_{max}}$, and $PPDW_{best} = \frac{FPS_{max}}{(T_{least}-T_a)\times P_{least}}$. $PPDW_{worst}$ is obtained when FPS generated is least (FPS_{least}) while the maximum power (P_{max}) is consumed and the device reaches the maximum peak temperature (T_{max}) allowed on the device; for example, generated FPS is 1 while executing all CPU and GPU cores at their corresponding maximum frequencies. On the other hand, the goal is to achieve the highest FPS possible with the least power consumed and least peak temperature achieved, which is denoted by $PPDW_{best}$ equation, where FPS_{max} is the maximum FPS, T_{least} is the least peak temperature achieved and P_{least} is the least power consumed; for example, $PPDW_{best}$ is obtained when 60 FPS is achieved while consuming least power with no rise in temperature. Fig. 4 shows the general trend of PPDW value as the FPS scales along with power consumption and peak temperature of the big CPUs achieved on Exynos 9810 MPSoC

[1]The overall temperature of the device is computed using the manufacturer's proprietor formula.

while executing Lineage 2 Revolution mobile game, which is a very computationally intensive game. In Fig. 4, the PPDW values (for FPS: 0, 1, 10) marked by red colour are the worst values achieved for the corresponding FPS while consuming the most power and achieving the maximum peak temperature of the big CPUs.

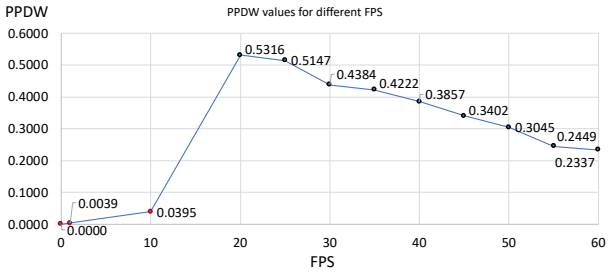

Fig. 4. PPDW value trend as the FPS, peak temperature of big CPUs and power consumption scale accordingly

$$optimize(PPDW) \rightarrow PPDW_{best} \geq PPDW_{desired} > PPDW_{worst} \quad (2)$$

IV. Proposed Methodology: Next

A. Overview of Next

Next is a software agent that executes continuously on the application layer of the Edge device employing MPSoC and runs on the most power efficient CPU, which is the LITTLE CPU of Exynos 9810 MPSoC, in order to consume the least power while executing. Since the agent runs on the application layer, no modifications to the existing hardware/software of the device is required.

The most important part of the Next methodology is to understand the user's interaction behaviour with the display/UI and its effect on the frame rate. To achieve this, the agent continuously monitors the frame rate every 25 ms for a window of *n* seconds. We call this virtual window of frame rates as *frame window*. From our empirical data we found that choosing the frame window for 4 seconds generates the best frame rate pattern analysis from user's interaction. Since, frame rate is usually denoted by frames per second (FPS), the agent has to scale the FPS accordingly due to monitoring of the frame rate every 25 ms. For 4 seconds of frame window we are able to capture 160 distinct values of frame rate during the user's interaction for that 4 seconds. The agent now computes the mathematical mode operation of all the 160 distinct values, which actually determines the most possible frame rate suitable to provide the desirable QoS for the user during that session.

Now, as shown in Fig. 5, the mode value is fed to the reinforcement learning (RL) module of the agent as the target FPS to achieve till the target FPS changes during the next frame window of user's interaction. The target FPS is now used for training purposes by the RL agent, and more details are provided in section IV-B. If we consider that the MPSoC consists of *m* number of processing element (PE) clusters and the current operating frequency of each cluster for cluster wise DVFS is denoted by f_i^{CM}, where $M \in 1, 2, ...m$ then the frequency values are fed to the RL module of the agent as part of the states. In our implementation, Samsung Note 9 (Exynos 9810 MPSoC) has 3 PE clusters namely: big CPU, LITTLE CPU and GPU. Once the agent is aware of the frequency states of the PE clusters along with power consumption, temperature, current FPS (referred to as $FPS_{current}$), which is the frame rate of the front buffer of VSync, and target FPS (referred to as $Target\_FPS$), the agent takes action to maximize reward, which in our case is to achieve the target FPS along with the best PPDW value. Once the training is complete based on the states and action values, the agent selects the desired operating frequencies ($f_{desired}^{CM}$, where $M \in 1, 2, ...m$) for the respective PE clusters and the *maxfreq* (maximum operating frequency) of each cluster is set to that desired operating frequency in order to achieve the target FPS and best PPDW for that FPS. Setting the *maxfreq* provides the flexibility for the PEs to operate within the range of maximum and minimum allowed operating frequencies.

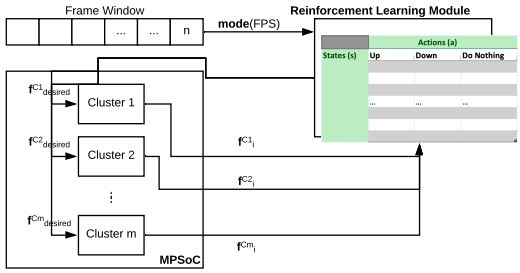

Fig. 5. Block digram of Next agent

B. Online Reinforcement Learning

Next is modeled to follow Q-Learning of reinforcement learning (RL) [24]. RL agent defines an environment (ϵ), in which the agent observes the state (s_i) at a time instance i and performs an action (a_i), and receives a reward (r_i) for that instance. At every time instance (i^{th}), the agent chooses an action a_i from a predefined list of actions with $a_i \in 1, 2, ...K$, where K is the maximum number of actions allowed for a given state. Following the action at time i any changes are perceived in the ϵ are observed at time $i + 1$, when the state of ϵ changes to s_{i+1}.

For our use case, if there are m number of PE clusters then for each cluster we would obtain 3 actions: Frequency up, frequency down, do nothing. Here we are under the assumption that each cluster of PEs only allow cluster wise DVFS (operating frequency is allowed for the cluster and not individual PEs). In this applicative case, we have 3 PE clusters on Exynos 9810 MPSoC and hence, there are 9 actions: big frequency up, big frequency down, do not change big frequency, LITTLE frequency up, LITTLE frequency down, do not change LITTLE frequency, GPU frequency up, GPU frequency down & do not change GPU frequency. It should be noted that setting operating frequency (up, down and do nothing) means to set the *maxfreq* of the respective PE (big, LITTLE, GPU) to that operating frequency. Setting the *maxfreq* to a particular value also means that the frequency is free to operate between the minimum allowed frequency (*minfreq*) of the PE cluster and the set *maxfreq*. In Next, the environment ϵ is defined by the states such as frame rate, power consumption and peak temperature of the Edge device running an application. For our Next implementation on Exynos 9810 MPSoC the following states are chosen as input: big_CPU_{freq}, $LITTLE\_CPU_{freq}$, GPU_{freq}, $FPS_{current}$, $Target\_FPS$, $Power_{current}$, $Temperature_{big}$ and $Temperature_{device}$[2]. Here, the value of $Target\_FPS$ is the mode of FPS values achieved from the *frame window*. Next is invoked every 100 ms to record the states and take actions.

The goal of the RL agent is to maximize reward r_i in the future. The propagation of information from the future

[2]$Temperature_{big}$ is the temperature of the big CPU cluster, whereas, $Temperature_{device}$ is the temperature of the overall device consisting of temperature of the battery and MPSoC, which could be captured from the device.

is discounted by a γ factor at every time step such that: $r_i = \sum_i^{i+n} \gamma_i r_i$ in order to dampen the rewards' effect on the agent's choice of action. For every time step the probability that the agent chooses an action at a given state is defined by a policy function. In this policy, the function which maximizes the agent's long term reward generation is called action-value function, which is defined by $Q(s_i, a_i)$ as shown in Eq. 3.

$$Q(s_i, a_i) = Q(s_i, a_i) + \alpha(r_i - Q(s_i, a_i) + \gamma max_a Q(s_{i+1}, a)) \tag{3}$$

In Eq. 3, α is the learning rate at which the agent learns new information. We have to keep in mind that the optimal action-value function could be obtained by iteratively updating $Q(s_i, a_i)$ in Eq. 3. Now, to maximize the reward generation we require a *reward function* ($R(s_i, a_i)$). For our *reward function* we use the Eq. 1 such that $R(s_i, a_i) = PPDW_i$. The agent's goal is to maximize reward, which means the agent has to optimize *PPDW* according to Eq. 2 and achieve $FPS_{current} = Target\_FPS$. The max reward generation could be represented using the following equation:

$$max\ R(s_i, a_i) = max(PPDW_i), \quad \text{where}$$
$$max(PPDW_i) = PPDW_{best} \geq PPDW_i > PPDW_{worst} \tag{4}$$

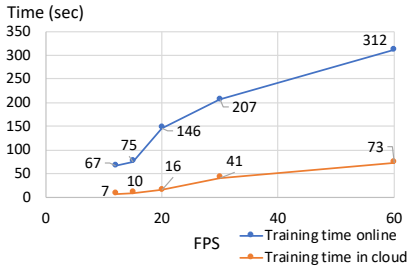

Fig. 6. Increase in training time for online vs cloud (offline) as the frame rate increases at part of the chosen states

For each application the training needs to be performed in order for the agent to make a learned decision when the application is executed by referring to the action-values. It should be kept in mind that if we consider all the possible value of frame rates (FPS 0 to 60) as part of the states and reward generation as mentioned above then the training time would be significant and hence quantizing the frame rate would be desirable for improved training time. Data series for *Training time online* in Fig. 6 shows the increase in average training time required for different frame rates for each application on the device. If we choose 60 frame rate then no quantization is required since that is the highest frame rate at 60 Hz refresh rate, whereas, for other frame rates we quantize the frame rate range. In our experimentation, choosing 30 frame rate results in the best training period on the Note 9 device. Since, through empirical data the average training period lasts around 3 minutes 27 seconds for a new application, which has not been executed/trained before, the agent achieved the best reward (PPDW) for the amount of time spent in training. Due to the Next agent executing on LITTLE CPU, the average power consumption during the training time does not exceed more than 6% of the average power consumption while executing the mobile application in general. The training for every newly executing application is only performed once and the Q-table (action-value) results are stored on the memory so that later when the application is executed again the agent is able to refer to the Q-table to set the correct frequency of different clusters (CPU/GPU). Given the training time is not significant compared to daily usage of mobile applications (4 hours 16 minutes), no offline training is required and the whole training could be performed on the device.

C. Offline training using Federated Learning or in Cloud

Given the fact that each Edge device manufacturers generally produce several different Edge device models, which are capable of executing similar mobile applications, a new type of machine learning called *federated learning* [25] could be utilized to train the agent more effectively by leveraging the computational power of the cloud. The training data from the Edge devices are sent back to the cloud server where the training of the agent happens. Once the training is complete the learned data (action-values) is sent back to the Edge devices, and hence, reducing the need of wasting local computing resource of the Edge device for training. Since cloud is computationally more powerful, the training period could significantly reduce to few seconds instead of minutes. Data series for *Training time in cloud* in Fig. 6 shows the reduced training time for different frame rate while the training is performed in a cloud system having Intel Xeon E7-8860V3 processor (16 cores) with 64 GB DDR3 RAM. Although it should also be noted that there was a maximum communication (to- and fro-) overhead of 4 secs between the device and the cloud system.

V. Experimental Results

Experimental setup: In this section, we compare the proposed Next methodology against the independent Linux CPU-GPU power management solution used in Android platforms by schedutil governor (we refer to it as *schedutil*) and QoS-aware power management methodology proposed by Pathania et al. [10] (we refer to it as *Int. QoS PM*). Due to lack in manufacturer's support and vendor locking it was not possible to install additional libraries in the Android kernel to access performance counters, and hence, we were not able to compare Next with other state-of-the-art methodologies such as [13], [14], which mostly rely on performance counter values to optimize power consumption or thermal behaviour. To evaluate the efficacy of Next we chose different types of applications from Google Play store [26] to get a more holistic view on power saving and reduction of thermal behaviour for such applications. From the pool of most popular applications we chose the following to represent different types of apps that a user would normally use: Facebook, Spotify music app, Chrome web browser (referred to as *Web Browser*), Lineage 2 Revolution gaming app (referred to as *Lineage*), PubG Mobile gaming app (referred to as *PubG*) and Youtube video streaming app. All the applications were used between 1 minute 30 seconds to 5 minutes per session, which is the general usage pattern by users according to [4], and the results reflect the average power consumption and peak temperature recorded during the sessions. For gaming applications (Lineage and PubG) each session lasted for 5 minutes, whereas, for other types of applications (Facebook, Spotify, Web Browser and Youtube) each session lasted between 1 minute 30 seconds to 3 minutes. For the results related to peak temperature observation, the experiments were all performed on the same day while the ambient temperature around the device was maintained around 21°C using controlled thermostat. All results for Next were observed when it was fully trained on the respective applications.

Power evaluation: Fig. 7 shows the average power consumption of Next, schedutil and Int. QoS PM for the chosen

aforementioned applications. For Next approach the power savings for Facebook, Lineage, PubG, Spotify, Web Browser and Youtube compared to schedutil are 37.05%, 50.68%, 40.95%, 32.98%, 32.11% and 40.6% respectively. Since, Int. QoS PM is used for power management for mobile games and the methdology could not be extended to all applications, we could only evaluate the methodology for Lineage and PubG apps. The power savings of Int. QoS PM for Lineage and PubG compared to schedutil are 16.31% and 23.84% respectively, making Next more power efficient by 41.07% and 22.47% respectively for the gaming apps when compared to Int. QoS PM, proving the effectiveness of Next in terms of power saving over the state-of-the-art power management approach such as Int. QoS PM.

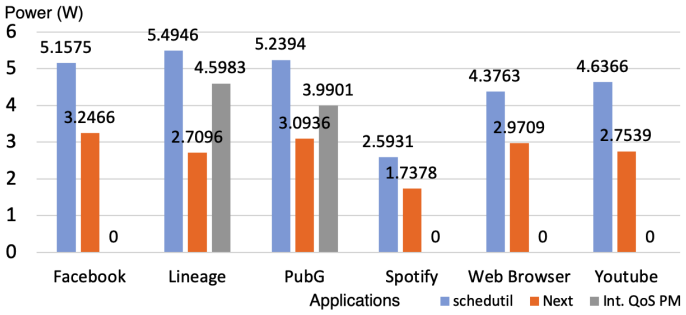

Fig. 7. Average power consumption for different mobile applications using schedutil, Next and Int. QoS PM approaches

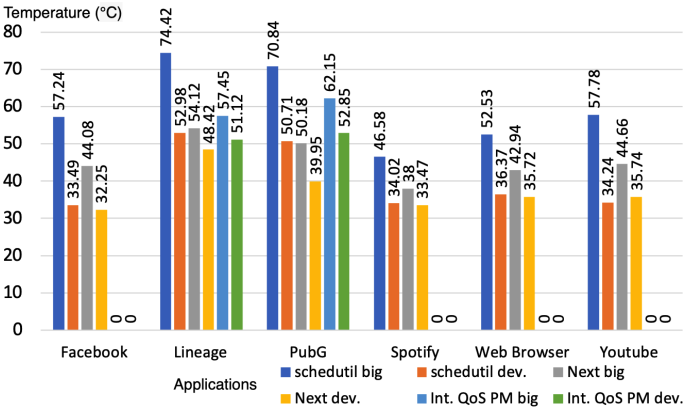

Fig. 8. Average peak temperature of big CPUs and the device for different mobile applications using schedutil, Next and Int. QoS PM approaches

Thermal evaluation: Fig. 8 shows the average peak temperature of big CPU cluster and the Samsung Note 9 device in general using schedutil, Next and Int. QoS PM. In the Fig. 8, *Int. QoS PM big* represents the average peak temperature of big CPU cluster using Int. QoS PM, *Int. QoS PM dev.* represents the average peak temperature of the device using Int. QoS PM, *Next big* represents the peak temperature of big CPU cluster using Next, *Next dev.* represents the peak temperature of the device using Next, *schedutil big* represents the peak temperature of big CPU cluster using schedutil, and *schedutil dev.* represents the peak temperature of the device using schedutil. From the Fig. 8 it is evident that comparing the results against schedutil Next is capable of reducing peak temperature by 29.16% (maximum) for big CPUs and 21.21% (maximum) for the device in general, whereas Int. QoS PM is only able to reduce the peak temperature by maximum of 22.80% for big CPU cluster and maximum of 3.51% for the device. This proves the effectiveness of Next over schedutil and state-of-the-art Int. QoS PM for its ability to reduce peak temperature of big CPUs and overall device.

Overhead analysis: From our empirical data, we noticed that the maximum overhead required for computation by the Next agent is around 227 ns on an average, which is computed over a session lasting for at least 5 minutes.

VI. Conclusion

In this paper, we propose a power and thermal efficiency agent for mobile MPSoC platforms based on reinforcement learning, which maximizes performance (FPS) while reducing power consumption and temperature of the mobile applications depending on the user's interaction with the display/UI and the desired QoS. Experimental evaluation on Exynos 9810 MPSoC shows the efficacy of the proposed approach along with its improvement over state-of-the-art power and thermal management scheme.

References

[1] "Top 5 stats to know about us mobile usage," https://www.emarketer.com/corporate/coverage/be-prepared-mobile.

[2] "How much time do we spend on social media?" https://mediakix.com/blog/how-much-time-is-spent-on-social-media-lifetime/.

[3] "Global mobile consumer survey: Us edition," https://www2.deloitte.com/us/en/pages/technology-media-and-telecommunications/articles/global-mobile-consumer-survey-us-edition.html.

[4] "Screen time stats 2019: Here's how much you use your phone during the workday," https://blog.rescuetime.com/screen-time-stats-2018/.

[5] "Galaxy note9," https://www.samsung.com/global/galaxy/galaxy-note9/, accessed: 2018-01-27.

[6] "Exynos 9 series (9810)," https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-9810, accessed: 2019-01-27.

[7] M. O. Johansson, M. Midholt, O. T. N. Moliner, and A. L. Hunt, "Adaptive touch panel synchronization," May 31 2016, uS Patent 9,354,744.

[8] B. D. Nilo, D. M. Chan, J. A. Xiao, and J. C. Beaver, "Devices and methods for processing touch inputs," Dec. 8 2016, uS Patent App. 14/870,879.

[9] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5. ACM, 2001, pp. 89–102.

[10] A. Pathania *et al.*, "Integrated cpu-gpu power management for 3d mobile games," in *2014 51st ACM/EDAC/IEEE DAC*. IEEE, 2014.

[11] O. Sahin *et al.*, "On the impacts of greedy thermal management in mobile devices," *IEEE ESL*, vol. 7, no. 2, 2015.

[12] R. A. Shafik *et al.*, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE TCAD*, vol. 35, no. 6, 2016.

[13] N. Peters *et al.*, "Frame-based and thread-based power management for mobile games on hmp platforms," in *2016 IEEE ICCD*. IEEE, 2016.

[14] G. Bhat *et al.*, "Online learning for adaptive optimization of heterogeneous socs," in *Proceedings of the ICCAD*. ACM, 2018.

[15] S. Dey *et al.*, "Edgecoolingmode: An agent based thermal management mechanism for dvfs enabled heterogeneous mpsocs," in *2019 VLSID*. IEEE, 2019.

[16] ——, "P-edgecoolingmode: An agent based performance aware thermal management unit for dvfs enabled heterogeneous mpsocs," *IET CDT*, 2019.

[17] S. Isuwa *et al.*, "Teem: Online thermal-and energy-efficiency management on cpu-gpu mpsocs," in *2019 DATE*. IEEE, 2019, pp. 438–443.

[18] S. Dey *et al.*, "Rewardprofiler: A reward based design space profiler on dvfs enabled mpsocs," in *2019 CSCloud/2019 EdgeCom*. IEEE, 2019.

[19] ——, "Deadpool: Performance deadline based frequency pooling and thermal management agent in dvfs enabled mpsocs," 2019.

[20] ——, "Socodecnn: Program source code for visual cnn classification using computer vision methodology," *IEEE Access*, 2019.

[21] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep q-learning model for energy-efficient edge scheduling," *IEEE TSC*, 2018.

[22] "Android 9 pie," https://www.android.com/versions/pie-9-0/, accessed: 2018-01-31.

[23] "Energy aware scheduling (eas)," https://developer.arm.com/open-source/energy-aware-scheduling, accessed: 2018-01-31.

[24] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[25] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[26] "Google play," https://play.google.com/store.