P-EdgeCoolingMode: An Agent Based Performance Aware Thermal Management Unit for DVFS Enabled Heterogeneous MPSoCs

Somdip Dey[1] ✉, *Amit Kumar Singh*[2], *Klaus Dieter McDonald-Maier*[3]

[1,2,3] *Embedded and Intelligent Systems Laboratory, University of Essex, Colchester, UK*
[1] *Samsung R&D Institute, UK*
✉ *E-mail: somdip.dey@essex.ac.uk*

Abstract: Thermal cycling as well as spatial and thermal gradient affects the lifetime reliability and performance of heterogeneous multiprocessor systems-on-chips (MPSoCs). Conventional temperature management techniques are not intelligent enough to cater for performance, energy efficiency as well as operating temperature of the system. In this paper we propose a light-weight novel thermal management mechanism (P-EdgeCoolingMode) in the form of intelligent software agent, which monitors and regulates the operating temperature of the CPU cores to improve reliability of the system while catering for performance requirements. P-EdgeCoolingMode is capable of pro-actively monitoring performance and based on the user's demand the agent takes necessary action, making the proposed methodology highly suitable for implementation on existing as well as conceptual Edge devices utilizing heterogeneous MPSoCs with dynamic voltage and frequency scaling (DVFS) capabilities. We validated our methodology on the Odroid-XU4 MPSoC and Huawei P20 Lite (HiSilicon Kirin 659 MPSoC). P-EdgeCoolingMode has been successful to reduce the operating temperature while improving performance and reducing power consumption for chosen test cases than the state-of-the-art. For applications with demanding performance requirement P-EdgeCoolingMode has been found to improve the power consumption by 30.62% at the most in comparison to existing state-of-the-art power management methodologies.

1 Introduction and Motivation

Modern embedded systems employ heterogeneous Multi-Processor Systems-on-Chips (MPSoCs), where several types of processing cores are available within a single chip, to deliver power as well as energy efficient computing. Some of the most popular heterogeneous MPSoC are the Samsung Exynos 5422 [1] and HiSilicon Kirin 659 [2]. Exynos 5422 employs 4 ARM Cortex A-15 (big) CPUs and 4 ARM Cortex A-7 (LITTLE) CPUs and 6 ARM Mali-T628 GPU cores, implementing ARM's big.LITTLE technology. Whereas, Kirin 659 employs 4 ARM Cortex A-53 (big) CPUs and 4 ARM Cortex A-53 (LITTLE) CPUs and 2 ARM Mali-T830 GPU cores. Now a days most of the research and development in MPSoCs have been focused on developing algorithm to provide energy efficiency while catering for performance and reduced thermal gradient [3–6]. But in practical sense, majority of the published algorithms lack the flexibility for real world implementation because the usage and workload on similar devices by different users are different and hence, requires the resource management methodologies to be flexible to adapt over time.

Elevated temperatures have adverse effects on Integrated Circuits (ICs) and reliability of electronic products can be heavily influenced by spatial or temporal gradients, or absolute temperatures [7]. To mitigate such reliability issues most mobile devices often come with thermal capping. Now a days devices often come with hardwired Thermal Management Units (TMUs) as well as software TMUs which regulates the energy consumption as well as the temperature of the associated components. But so far none of these state-of-the-art thermal management units have been proven to be effective enough (see Sec. 2) to cater for performance, energy efficiency as well as thermal-regulation.

The Exynos 5422 and Kirin 659 SoCs also support Dynamic Voltage Frequency Scaling ($DVFS$) capabilities, which could be used to reduce dynamic power consumption ($P \propto V^2 f$) [8–10]. DVFS helps to reduce the energy consumption by executing the workload over extra time at a lower voltage and frequency, which could be accounted for reduced power consumption. In order to cater for performance several resource mapping and partitioning mechanisms using DVFS [3, 4, 8, 11–13] have been proposed while keeping energy consumption low. Since applications can be classified into three categories [9]: compute intensive, memory intensive and mixed load (both compute and memory intensive), given most applications in real-world fall under the mixed load category, we had the following observations:

Observation 1: ARM Cortex A-15 CPU being an out-of-order sustained triple-issue processor, is capable of providing maximum performance, but when we ran a mixed load program on such CPU, the core temperature rises very fast due to executing workload that are both compute and memory intensive.

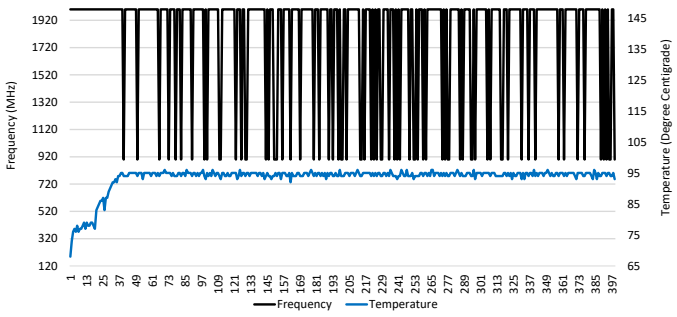

Fig. 1: Frequency and Temperature vs Execution Time Comparison of Streamcluster

Fig. 1 shows the motivation to design a dynamic thermal manager for executing applications on a heterogeneous multi-core architecture (Exynos 5422) containing two types of cores in clusters

(big.LITTLE), where 4b and 4L cores are present. The horizontal axis shows the execution time steps whereas the vertical primary axis (left-hand side) reflects the Frequency (in MHz) for the big cores on which the program is executing and the vertical secondary axis reflects the temperature (in °Centigrade) of the big core, which has the worst temperature behavior, while executing the workload. Usually the cores exhibit worse thermal behavior which are residing close to the memory. We chose Streamcluster* from the PARSEC benchmark suit [14] to be our mixed workload on the big cluster. The benchmark completed its execution in 433.12 secs, but Fig. 1 only shows the variance result for first 39.9 secs of the execution because there was repetition in the behavior of the results. From the figure, it could be noticed that the peak temperature† on the big core was around 95° centigrades and because of this the thermal management unit (TMU) of the OS starts to regulate the temperature of the cores by CPU throttling‡. For none of our experiments, we could profile the temperature behavior of LITTLE cores on the Exynos 5422 not just because they are less powerful (1/4 size of the big ones and operates at a lower frequency) but there is no temperature sensor for LITTLE cores onboard as well. Therefore all our results only focused on thermal behavior and CPU throttling on the big cores on Exynos MPSoC. Although CPU throttling is good in terms of reliable operations but the stock thermal management algorithm of the OS is not intelligent enough to provide performance at the same time. In Fig. 1 we could notice that the clock speed varies from 2000 MHz to 900 MHz at a periodic time step, which causes a drop in performance of the executing application(s). Thus it is necessary to design a thermal manager, which is intelligent enough to cater for performance as well as maintain a desirable operating temperature of the cores.

Observation 2: In a study [9] by Basireddy et al., the researchers have proposed a novel workload management system, which classifies workloads of the executing applications based on Memory Reads Per Instruction (MRPI) metric and manages DVFS levels of cores based on it. This method has resulted to 33% more energy efficiency as compared to modern state-of-the-art workload management approaches and Fig. 2 shows that efficient workload management could also result in energy efficiency as well as performance. For this experimentation, we executed the same Streamcluster benchmark, which completed its execution in 409.596 secs, but Fig. 2 only reflects the variance result for first 39.9 secs of the execution because there was repetition in the behavior of the results. Another interesting thing that could be noticed is that the peak temperature of the big cores also reduced when the workload was mapped appropriately between the CPU cores executed with appropriate DVFS levels for similar mixed-load applications. In some studies [15, 16] it has been found that by reducing the operating temperature by 10-15° centigrades could improve the lifespan of the device by 2x. Since the reliability of the device is highly dependent on the operating temperature of the device, devising temperature-aware mechanisms capable of reducing the operating temperature can help improve the life-span of the SoC device. Therefore, there is a desperate need to implement dynamic thermal manager, which is capable of regulating the operating temperature of the system on top of the state-of-the-art workload and resource management mechanisms so that we could cater not just only energy efficiency and performance but also the overall reliability of the MPSoC.

In our previously published methodology, EdgeCoolingMode [6], we are able to reduce the operating temperature of the device by 6.32% (average) while improving the performance by 7.16% compared to Linux's Ondemand governor and reduce the operating temperature of the device by 6.32% (average) while improving the performance by 7.85% compared to Linux's Performance governor while executing Streamcluster benchmark (See Sec. 4 for detailed analysis). This goes on to prove that if EdgeCoolingMode methodology [6] is aware of performance/throughput pro-actively while reducing thermal gradient then there is a possibility of reducing the operating temperature even more.

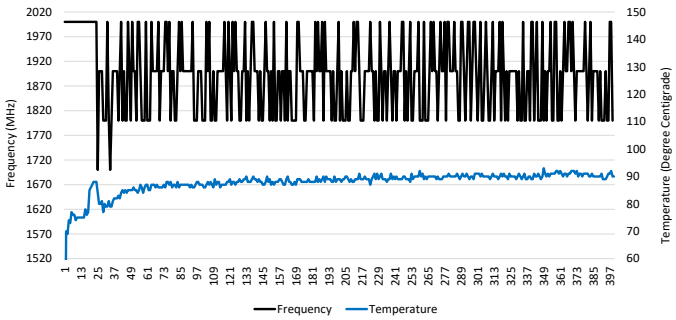

Fig. 2: Frequency and Temperature vs Execution Time Comparison of Streamcluster Using MRPI

In order to overcome the limitations of the existing approaches (as mentioned in Observation 1 and 2) towards addressing temperature-aware resource management mechanisms, which are capable of catering for both performance and reduced energy consumption, we propose a novel light-weight dynamic thermal management mechanism using intelligent software agents based on a fast heuristic approach. This performance aware thermal management mechanism could be used as a module sitting in the application layer to pro-actively monitor operating temperature and performance of the executing applications. We call our proposed methodology, "P-EdgeCoolingMode", since the intelligent software agents monitors and regulates the operating temperature of the system while taking throughput in to consideration. To this end, this paper makes the following contributions:

1. A light-weight mechanism in the form of **intelligent software agent to monitor and regulate temperature of the CPU cores** to improve reliability of the system while pro-actively monitoring performance.
2. **Validation** of the proposed thermal management intelligent agent on real hardware platforms such as the **Odroid-XU4** [17] and **Huawei P20 Lite** [18].
3. To show the **efficacy and flexibility of our proposed methdology (P-EdgeCoolingMode)** we have implemented it on **Ubuntu OS** as well as on **Android OS** and highlighted the experimental results in Sec. 4.

The rest of the paper is organized as follows. Section 2 presents the state-of-the-art work pursued in the same field. Section 3 describes our proposed methodology and its implementation. Section 4 shows the system model describing the hardware and software infrastructures used for our experiments, the experimental results and validation of our proposed approach. In Section 5 we explore some related discussion on the proposed methodology and finally, Section 6 concludes the paper.

2 Related Work

In several earlier studies, many researchers have focused on designing methodologies and frameworks to optimize power and operating temperature of MPSoCs. One such noteworthy study is performed by Ghasemazar et al. [19] where the researchers proposed a hierarchical framework leveraging DVFS capabilities of the processing cores to find the optimal voltage-frequency to cater for power consumption and temperature. This methodology was successful in achieving 20% performance boost without impacting the overall operating temperature but their experiments focused mainly on CISC architectures and

*We chose the native option of the Streamcluster to mimic real-world data-mining algorithms, which are both compute intensive and memory intensive.

†For our study we have fetched the peak temperature out of the 4 big cores on the Exynos 5422 SoC. More details are provided in Section 4.

‡Adjusting the clock speed of the CPU to use less energy consumption and reduce CPU temperature for improved reliability.

all results were based on a MATLAB-based Chip Multiprocessor simulator. In another paper [20] Kamal et al. proposed a heuristic based thermal stress-aware mechanism for management of power and temperature in MPSoCs formulated in a convex optimization problem. This approach was implemented on Sniper multicore simulator [21] and was able to reduce spatial and temporal thermal gradients by 7% and 18% respectively when compared to the work in [19]. In another work by Iranfar et al. [22], the researchers proposed a multi-tier hierarchical thermal stress-aware power and temperature management framework for MPSoCs, where the methodology used similar convex optimization solution in multi-layer to improve mean time to failure (MTTF)*. The effectiveness of this methodology is again proved based on simulations performed on Sniper multicore simulator. All the aforementioned noteworthy studies were implemented and experimented in simulations instead of experimenting on real devices, which could differ a lot from the practical results achieved from real devices because there are so many factors that account for the operating temperatures such as ambient surrounding temperature, chemical reactions on the devices due to weather change, etc. Although modern simulators such as Sniper multicore provides simulation results very close to real devices but there are many unaccountable factors that could happen when executed on a real device instead of simulation, especially in case of temperature variance.

In [23] Sigla et al. present a predictor using power sensors to predict the next power consumption based on the following frequency setting is developed. Their technique uses a leakage power model of the ARM's big.LITTLE architecture on the Odroid-XU3 to validate its predictor and Dynamic Power and Frequency Management technique. An Extension of this work has also been published in [24]. Both [23, 24] methods involve predicting the future core temperatures to adjust the workloads or frequencies before exceeding a set threshold, but computation of such predictive temperatures increases the performance overhead of such methods. Another issue with such methodology is that predicting future core temperatures only work with applications consisting of periodic tasks, where the workload of the task is predictable in the future. Whereas, most real world (real-time) applications [25–27] on mobile devices such as social media, email composer/reader, music player, etc. which also rely on different types of operations and are not predictable, consist of aperiodic and sporadic tasks as well. A sporadic task [26] is released at random time instants and have hard timing constraints, whereas, aperiodic task invocation is unknown at design time, making both sporadic and aperiodic tasks difficult for future prediction.

To solve the aforementioned issue of predicting thermal behavior and power consumption of real world applications, Reinforcement Learning (RL) could seem to be a viable option at first, but such approaches have their own disadvantages. In the study [28] by ul Islam et al., the authors proposed Reinforcement Learning based DVFS approach to improve energy consumption of the device, but the proposed methodology is based on Q-Learning where the states and actions are noted in a Q table. Since the methodology learns from the actions it takes and what kind of reward or penalty it receives from taking such an action by updating the Q table, it might take the approach some time to reach an optimized solution, which is optimized energy-consumption in this case. Not to mention that in a resource constrained device where the total amount of available memory on the device is restricted or limited utilizing Q learning could be a challenge. Additionally, the methodology does not proactively control the thermal behavior of the device. In another study [27] by Zhang et al., the authors proposed a Reinforcement Learning and Deep Learning based DVFS selection mechanism, where based on the periodic tasks the appropriate DVFS algorithm is chosen. But the issue with this study is that it only workds for periodic tasks where sequence of tasks are predefined and easy to predict. Moreover, if a new type of task is introduced in the task graph then re training of the mechanism is required, which makes it unfavorable for applications with ever changing task set. In both the aforementioned studies [27, 28], efficacy of the methodologies are validated on simulation platform with generated task sets that do not represent application with sporadic and aperiodic tasks similar to usage of mobile platform users.

In a different study [29], the authors propose a deep Q-learning methodology for dynamic thermal and power management. This approach takes less memory due to the reduced number of bins and hence reducing the number of rows in the Q-table. However, in this approach the Q-table still has 15 states (sum of little core utilizations, big and little core frequencies, number of big and little cores, total power consumption, and five normalized performance counters) and 3 actions making the memory consumption to store the table high. Although the experiments were performed on Odroid XU3 platform, the approach still depends on profiling of application during design time to be fed for training to produce best rewards. Moreover, the approach is not automated in nature and hence, reduces viable application in commercial devices.

In our previously published work, EdgeCoolingMode [6], we have developed a thermal management mechanism, which is capable of pro-actively monitoring thermal behavior of the device while addressing all the limitations faced by the aforementioned related published works. But for commercial mobile devices, where Quality of Service is critical for user satisfaction, performance requirements become more important than thermal behavior or energy consumption on the device itself for certain applications. To be able to pro-actively monitor performance and temperature to satisfy both performance requirement and reduced thermal gradient, based on the need of the user we introduced P-EdgeCoolingMode, an agent based performance aware thermal management unit. In our current approach (P-EdgeCoolingMode), we utilize the concept of intelligent software agents and linear regression based supervised machine learning to design a light-weight fast heuristic based operating temperature regulator, which could be used on top of any scheduler or governor or any other types of system resource manager to achieve reduced operating temperature while catering for desired performance and improving energy efficiency in the long run.

3 Proposed Methodology: P-EdgeCoolingMode

3.1 Overview of P-EdgeCoolingMode

There are four schools of thought for artificial intelligence [30]: Thinking Rationally [31], Acting Rationally [32], Thinking Humanly [33] & Acting Humanly [34], where intelligent agents are built or designed to portray each school of thought. In our P-EdgeCoolingMode we propose an intelligent software agent that would monitor and regulate the thermal behavior of the system by *Thinking and Acting Rationally* based on our heuristic approach, and operates on top of the existing scheduler of the system.

Fig. 3 reflects the working of P-EdgeCoolingMode, which starts by defining a thermal budget by the user and a performance deadline (referred to as performance threshold or P_{max} symbolically). The performance deadline could be optional and if no value for this parameter is provided to P-EdgeCoolingMode then by default the agent just uses the thermal budget to reduce thermal gradient. P-EdgeCoolingMode agent has two distinct modules: Learning module, which monitors the operating frequency, operating temperature of the cores and performance of the executing application and creates a relationship variable (α) based on linear regression between operating frequency and operating temperature. Although relationship between power and temperature [7], therefore frequency ($P \propto V^2 f$) and temperature does not follow a linear relationship in practice but to make our heuristic approach fast we make the assumption that the relationship is linear and try to reduce the error capacity of our approach by defining an error variable (ε). Most of the time the default scheduler of the system already considers the task utilization and maximum capacity of the processing elements (CPU) to decide the best possible frequency to operate the task and since, P-EdgeCoolingMode executes on top of existing system scheduler, we have only considered to use the operating frequency and thermal

*Mean time to failure (MTTF) is the length of time a device is expected to operate/last till failure.

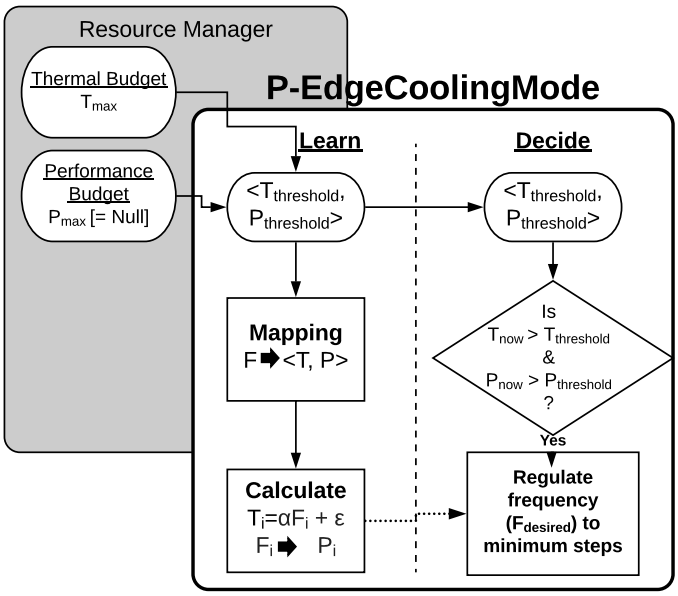

Fig. 3: P-EdgeCoolingMode Software Agent

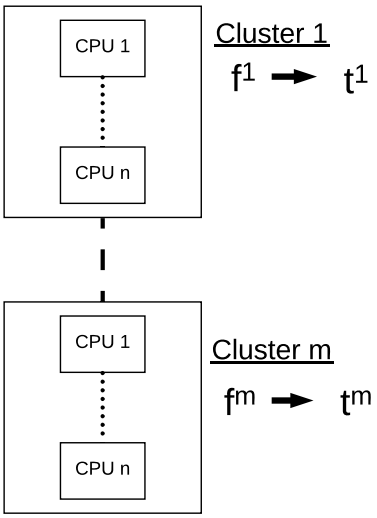

Fig. 4: Cluster wise frequency scaling and cluster wise thermal behavior

behavior of the processing elements to deduce the linear relationship for faster execution and practical implementation.

Based on the user-defined maximum threshold of the operating temperature of the system, the Learning module updates the value of the relationship (α) and error (ε) variables while recording the performance of the application at the same time. In the Decision module, if the performance deadline is not defined then the agent adjusts the operating temperature of the system by deciding by how much the operating frequency needs to be reduced based on the value of α deduced in the Learning module. In contrary, if the performance deadline is defined then the agent caters for the performance by selecting the appropriate operating frequency based on the mapping created between operating frequency and performance in the Learning Module. In order to make our P-EdgeCoolingMode agent more intelligent and adaptive, we have also adopted the concept of $Continuous Learning$ [35–37], which is bio-inspired. A human-being continuously learns from its environment and tries to adapt to the surrounding through continuously learning from it. We replicate the same concept in our P-EdgeCoolingMode agent so that with changing environment and conditions, the agent could adjust the α and ε variables along with the performance achieved for an application on a particular operating frequency. Our proposed approach can work with the existing Governor or dynamic thermal and power management (DTPM) of the OS or any other resource manager that is installed on the system or is working in co-habitation. For majority of the proposed work [19, 20, 22, 23, 27, 28] in this area require either modification in the kernel of the OS or heavy modification to the existing infrastructure of the devices in order to be implemented, however, our agent could be deployed in the application layer without rebuilding the kernel or without making modification to device's infrastructure, making it easy to implement on existing devices and in devices to come in the future.

3.2 Learning Module

The algorithm for Learning module is provided in Algo. 1.

In the study [7], through experiments on real devices it was found that on heterogeneous MPSoCs such as Odroid platforms, power and temperature follows a relationship of quadratic function or exponential function based on various other dependable factors*. We also validate this relationship in Sec. 5. However, to make our heuristic approach fast during run time to take necessary decision for thermal regulation we assume that frequency and temperature follows a linear relationship represented by Eq. 1. In Sec. 4 also show the effectiveness of using such linear regression based methodology, which has yielded amazing results in terms of performance and reduction in temporal thermal gradients considerably.

$$\begin{aligned} T_i &= \alpha \times F_i + \epsilon \,, \\ where\ F_i &: \text{Operating frequency at time instance i,} \\ T_i &: \text{Operating temperature at time instance i,} \\ \alpha &: \text{Relationship variable,} \\ \epsilon &: \text{Error variable} \end{aligned} \tag{1}$$

In the Learning module, the agent monitors the thermal changes using Eq. 1 and evaluate the value of α and ϵ based on Eq. 2. If we consider that α and ϵ remain constant between two temperature variance and, F_2 and T_2 are the operating frequency and operating temperature respectively at the time instance, whereas F_1 and T_1 are the operating frequency and operating temperature respectively at previous time instance then we could derive:

$$\begin{aligned} T_2 &= \alpha \times F_2 + \epsilon \ , \\ T_1 &= \alpha \times F_1 + \epsilon \ , \\ \therefore \alpha &= (T_2 - T_1)/(F_2 - F_1) \end{aligned} \tag{2}$$

From Eq. 2 we could solve for α. Thus, for every instance of operating frequency of the cluster, i.e. F_i, the agent maps it to a peak operating temperature of the cluster cores (T_i) (see Eq. 3). Since, in this study, we have focused on hardware platforms such as Exynos 5422 [1] and Kirin 659 [2], which only allows cluster wise DVFS, hence, P-EdgeCoolingMode agent regulates the operating frequency (F_i) of the cluster to monitor the operating temperature (T_i) of the cluster. In Fig. 4, we show cluster wise DVFS mechanism. If we consider that a MPSoC consist of m number of clusters of processing cores, where each cluster consists of n number of processing cores, then for each cluster we could have an operating frequency (f^i) which leads to the operating temperature (t^i) of the cluster. Hence, m different clusters we could have different operating frequencies represented as $\{f^1, f^2, ...f^m\}$ and different operating temperature $\{t^1, t^2, ...t^m\}$ associated with these frequencies. For simplicity of

*Power/Temperature relationship could vary because of whereabouts of the temperature sensors onboard or distance between temperature sensors and hotspot, etc.

understanding the implementation of the agent, let us generalize the operating frequency and operating temperature as F_i and T_i respectively.

$$F_i \longmapsto T_i \tag{3}$$

Now from Eq. 3 and 2 and by reducing the frequency (using DVFS capabilities) by a certain number of frequency scaling levels*, different F_i, T_i, α, ϵ, frequency scaling level reduction steps (l) along with the performance (P_i) of the application for the respective F_i are recorded by the agent, which would be used to decide which frequency to drop to in the Decision module when the operating temperature reaches the threshold value†. We have to keep in mind that an instance of operating frequencies (F_i) do not just lead to an instance of operating temperature (T_i) but also leads to an instance of performance (P_i) achieved. Hence, P-EdgeCoolingMode maps the operating frequency to a tuple consisting of operating frequency and performance of the executed application, represented by Eq. 4, which is an extension of Eq. 3.

The readings of the values such as F_i, T_i, P_i (refer to *Mapping Frequency with Temperature* in Algo. 1) are kept on the fast volatile memory such as RAM for fast access by P-EdgeCoolingMode, while a copy of the values are also recorded in a file, which is saved in *map.txt* file on the non-volatile memory such as hard-disk/external storage so that the values could be fetched (refer to *Calculate* α & ϵ in Algo. 1) at any point even after the execution of the application has completed.

Note: Performance could be defined by the user based on the type of application being executed. For some applications where Quality of Service (QoS) is measured by how fast the application is computed in that case performance would be the execution time of the application, whereas for some other application where performance of a game could be evaluated based on the rendered frames per second (FPS) in that case the performance would be defined as FPS for the gaming application.

$$F_i \longmapsto < T_i, P_i > \tag{4}$$

3.3 Decision Module

The algorithm for this module is provided in Algo. 2.

In the decision module, if the performance budget i.e. performance deadline (performance is set to Null as variable in the implementation) is not mentioned then by default P-EdgeCoolingMode optimizes for operating temperature only without actively meeting performance deadline by regulating the operating frequency based on Eq. 3. The P-EdgeCoolingMode agent uses different relationship variable (α), error variable (ϵ), frequency scaling level steps l_i computed from Algo. 1 and then calculate the desired frequency using the following equation:

$$F_i - F_{desired} = (T_i - T_{desired})/(\alpha) \ , \quad where \ T_{desried} < T_{threshold} \tag{5}$$

The Eq. 5 is deduced from Eq. 3 and 1, and later verified under different experimental setup on different devices with different processing capabilities to be effective (see Sec 4) to reduce the thermal gradient and power consumption without hurting the performance. The P-EdgeCoolingMode agent tries to find the least value of $F_i - F_{desired}$ i.e. the least number of frequency scaling level, it should drop to so that the operating temperature could be reduced without affecting the overall performance of the executing application as opposed to what generic TMUs or DPTMs‡ do. Majority of stock TMUs and DPTMs reduce the frequency of the cores drastically to achieve thermal and power budget but that also reduces the performance of the executing application drastically.

When performance deadline is not defined then P-EdgeCoolingMode does not monitor performance directly but since for most computational application operating frequency is directly proportional to performance and hence by reducing the frequency by least scaling level the agent not just reduces the operating temperature but also try to affect performance in a passive way. Therefore, from the Eq. 5 the agent has to find the least value of $F_i - F_{desired}$ by computing the predictive operating temperature using the values of α_i & ϵ_i for every frequency scaling level steps (l_i). In the Eq. 6, we could notice that using Eq. 6 the agent would be achieving several possible frequency scaling levels to drop to but the agent has to choose the least value from the set to ensure performance is affected the least.

$$\forall\{l_i \in n\} : l_{desried} = (F_i - F_{desired})_{least} \ , \quad where \ l_i : \text{each frequency scaling level} \tag{6}$$
$$n : \text{number of different frequency scaling level steps}$$

Now, when the performance deadline is defined by the user in the P-EdgeCoolingModefor an executing application then from the operating frequency and achieved performance mapping deduced from Eq. 4 in the Learning module, the operating frequency, which is able to satisfy the defined performance deadline, is chosen from the mapping table instead of using the relationship deduced from Eq. 5.

Algorithm 1: Learning Module Execution

Input:
1. T_{max}: threshold value of operating temperature
2. P_{max}: threshold value of performance for the particular executing application
3. n: number of different frequency scaling levels

Output: $S(\alpha, \epsilon, l)$: set of α values for s frequency scaling levels

Initialize:
$T_{threshold}$ = T_{max};
$P_{threshold}$ = P_{max};

Mapping Frequency with Temperature:

Write(F_i, T_i, P_i, map.txt); ▷ Monitor and track different frequency, temperature readings & performance (Eq. 3)

Calculate α & ϵ:
Read(map.txt); ▷ Monitor map.txt file for changes in T_i
if $T_{i-1} \geq T_{threshold}$ **then**
  for *each frequency scaling level* l_i *in* n **do**
    $< \alpha_i, \epsilon_i >$ = CalculateAlphaEpsilon(l, F_i, T_i, F_{i-1}, T_{i-1}); ▷ Compute α & ϵ using the Eq. 2
    Write($S(\alpha_i, \epsilon_i, l_i)$, alphas.txt);
      ▷ Write our α, ϵ & l values so that it could be later updated through conituous learning return $S(\alpha_i, \epsilon_i, l_i)$;
else
  return void();

*For Exynos 5422 [1] big core cluster has 19 frequency scaling levels with 100MHz each step, whereas the Kirin 659 [2] big core cluster has 5 frequency scaling levels.

†*Here, temperature threshold is the thermal cap of the CPU cores that the P-EdgeCoolingMode agent regulates such that* $T_i \leq T_{threshold}$.

‡*DPTM is Dynamic Power and Thermal Management techniques that regulates both power and temperature.*

It should be kept in mind that the α values deduced from the Learning module are not just saved on the fast volatile memory such as RAM, but also saved in a file (*alphas.txt*) on the non-volatile memory such as hard disk or external storage so that the values could be fetched even after the application has exited or stopped executing (refer to *Decide* in Algo. 2). The values of F_i, T_i, P_i recorded in the Learning module are also stored on the volatile memory for fast access as well as on a file (*map.txt*) on the non-volatile memory for later access.

Algorithm 2: Decision Module Execution

Input:
1. T_{max}: threshold value of operating temperature
2. P_{max}: threshold value of performance for the particular executing application
3. n: number of different frequency scaling level steps

Output: $(F_i - F_{desired})_{least}$: least desired operating frequency to drop to

Initialize:
$T_{threshold}$ = T_{max};
$P_{threshold}$ = P_{max};

Decide:

```
if P_max == Null then
    S(α_i, ε_i, l_i) = Read(alphas.txt);
                          ▷ Read the α_i and ε_i for each l_i
    if T_i ≥ T_threshold then
        for each frequency scaling level l_i in n do
            l_this = CalculateLeastFrequency(α_i, F_i, T_i,
              F_{i-1}, T_{i-1});
                  ▷ Compute l_this, which is (F_i − F_desired)
            if l_this ≤ l_i && l_this ≥ l_prev then
                SetOperatingFrequency(F_desired);
                  ▷ Set the operating frequency to the desired
                  optimal one
                return (F_i − F_desired)_least;
            l_prev = l_i;
    else
        return void();
else
    < F_i, T_i, P_i > = Read(map.txt);
    Select F_i as F_desired where P_i ≥ P_max;
```

3.4 Continuous Learning

The Learning module keeps running and keeps updating the value of relationship variable (α), error variable (ϵ) and performance achieved for different associated frequency scaling levels for which the operating frequency should be reduced to reduce spatial and temporal thermal gradient.

4 Experimental Results

4.1 System

4.1.1 Hardware 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 Multicore Processors (AMPs) system-on-chip (AMPSoC), which is a special case of heterogeneous MPSoC and has clustered cores on the system. Our study was pursued on two different MPSoC platforms:

1. The Odroid XU4 board [17], which employs the Samsung Exynos 5422 [1] MPSoC.
2. The Huawei P20 Lite [18] smart-phone, which employs the HiSilicon Kirin 659 [2] MPSoC.

Exynos 5422 MPSoC: Exynos 5422 is based on ARM's big.LITTLE technology [38] and contains cluster of 4 ARM Cortex-A15 (big) CPU cores and another of 4 ARM Cortex-A7 (LITTLE) CPU cores, where each core implements the ARMv7-A ISA. This MPSoC provides dynamic voltage frequency scaling feature per cluster, where the big core cluster has 19 frequency scaling levels, ranging from 200 MHz to 2000 MHz with each step of 100 MHz and the LITTLE cluster has 13 frequency scaling levels, ranging from 200 MHz to 1400 MHz, with each step of 100 MHz. Additionally, each core on the cluster has a private L1 instruction and data cache, and a L2 cache, which is shared across all the cores within a cluster.

Since Odroid XU4 board does not have an internal power sensor onboard, hence an external power monitor [39] with networking capabilities over WIFI is used to take power consumption readings. Although the ARM Cortex-A7 (LITTLE) CPU cores on Odroid XU4 do not have temperature sensor but our intelligent agent approach is scalable and works for heterogeneous cluster cores.

Kirin 659 MPSoC: Kirin 659 is also based on ARM's big.LITTLE technology and contains a cluster of 4 big CPU cores and a cluster of 4 LITTLE CPU cores. But the big.LITTLE implementation of this MPSoC is unique, since it uses the same type of CPU core for big as well as LITTLE. Kirin 659 MPSoC uses Cortex A-53 as both big and LITTLE CPU cores, which implements ARMv8-A ISA, supporting 64 bit instruction set and is userspace compatible with 32-bit ARMv7-A architecture. Similar to Exynos 5422, this MPSoC also provides dynamic voltage frequency scaling feature per cluster, where the big core cluster has 5 frequency scaling levels ranging from 1402 MHz to 2362 MHz (at the following frequencies: 1402 MHz, 1805 MHz, 2016 MHz, 2112 MHz, 2362 MHz), and the LITTLE core cluster has 4 frequency scaling levels ranging from 480 MHz to 1709 MHz (at the following frequencies: 480 MHz, 807 MHz, 1306 MHz, 1709 MHz).

Huawei P20 Lite smart-phone has power sensors as well as 13 thermal sensors, but due to lack of documentation from the vendor on the positioning of the thermal sensors it is not feasible to associate the installed temperature sensors with specific cores/cluster. For this reason, the experiments on Kirin 659 focused on the performance of the executed applications and associated power consumption of the device.

4.1.2 Software Infrastructure:

4.1.3 Experiments on Exynos 5422: For multi-core systems, multi-threaded applications are heavily used in recent times to represent workloads as they could leverage concurrency and parallel processing. Examples of such applications are available in several benchmarks such as PARSEC [14]. For our experiments we have tried several applications from the PARSEC benchmark such as Streamcluster, Facesim, x264, etc. but to validate the effectiveness of our $P - EdgeCoolingMode$ mechanism we chose Streamclsuter with $native$ option because it closely represented a real-world mixed load application and the execution period was long enough to observe thermal regulation in the system. We also validated our approach for Whetstones benchmark [40]. We have run all these applications on the Exynos 5422 MPSoC having UbuntuMate OS version 14.04 (Linux Odroid Kernel: 3.10.105).

4.1.4 Experiments on Kirin 659: To emulate the usage of mobile applications by users on real mobile devices, instead of choosing benchmark applications such as PARSEC for experimental setup, we chose mobile applications that could be used to consolidate a typical user's behavior on the device itself. For this reason, we chose the following common actions that consolidate the behavior of users on Android:

- **Downloading** an app using the **Google Play Store** application.
- **Streaming** Youtube videos on **Chrome** web browser application.
- **Idle** background processes.
- **Taking pictures/recording videos** on **Camera** application.

Since, the performance of the aforementioned actions on an Android is mostly related to rendered frames per second (FPS) because of Vertical Synchronization (VSYNC) [41]. VSYNC is basically the refresh rate of the display. Whenever we perform an action on Android, every action updates the smart-phone's display to return something viewable to the user. If the result of an action is not updated on the display before the next VSYNC update time then the user experiences tearing effect on the display because of dropped frames. Henceforth, in all our experiments on the Kirin 659 MPSoC we are more focused on FPS for the aforementioned actions as the performance parameter for P-EdgeCoolingMode. We have executed the aforementioned actions on the Kirin 659 MPSoC running on the Android 8.0.0.168 Oreo OS (Linux Kernel: 4.4.23+ #1 SMP).

4.2 Experimental Setup

4.2.1 Experiments on Exynos 5422: We implemented our proposed P-EdgeCoolingMode software agent on top of the MRPI based Mapping and Resource Manager [9] as well as on stock Linux Governors to perform the experiments and validate the effectiveness of our methodology. For all our experiments we did not modify anything else on the system, hardware or software, so that the agent's standalone effectiveness could be determined. We choose 7 different operating temperature thresholds: 95°, 94°, 93°, 92°, 91°, 90°and 89°.

4.2.2 Experiments on Kirin 659: As mentioned in Sec. 4.1.4, we performed the following actions on Huawei P20 Lite: downloading an app from Google Play store, streaming Youtube on Chrome, idle mode where no major application runs in foreground and taking picture/recording video. To reproduce the same task set, we extracted the related task set from the task tree on the device and executed the same task set with same CPU affinity to evaluate a fair comparison between P-EdgeCoolingMode and default governor of Android based Linux.

4.3 Experimental Results

4.3.1 Experiments on Exynos 5422: In table 1 we provide the different α and ϵ values for different threshold (90-95°)* for drop of frequency by two-step levels (reduction of 200 MHz). Here each α and ϵ are the values for the instance when operating temperature reached the threshold value and the agent reduced frequency by two-steps to reduce the operating temperature.

Fig. 5 and 6 show the operating temperature & frequency respectively of the big core, which was monitored during the execution of benchmarks with our proposed P-EdgeCoolingMode agent in effect as well without utilizing the agent. We choose to show only values for 5 temperature threshold: 89°, 91°, 93°& 95°, so that it could be easily understood from the graph.

We ran Streamcluster from PARSEC five times each for different temperature threshold values (89°, 90°, 91°, 92°, 93°, 94°& 95°) and for MRPI [9], Linux governor in performance mode, Linux governor in ondemand mode & Deep Q-Learning Dynamic Management [29]. Fig. 7 summarizes the average execution time (in secs), average peak temperature (in ° centigrade), average power consumption (in Watt) and the average number of times CPU throttling took place. The average is computed by taking all 5 execution for each temperature budget into account. From the table in Fig. 7 it could be noticed that using our P-EdgeCoolingMode we have achieved 6.84% reduction in peak temperature† while improving performance by 7.16%

*We only showed from 90-95° centigrades for this experiment to show the computations of α and ϵ for space constraint in this paper.

†Considering that Linux's thermal cap is at 95°centigrades.

Table 1 Results: Values of α & ϵ for different temperature thresholds

95°	94°	93°	92°	91°	90°
$\alpha = 0.02$, $\epsilon = 55$	$\alpha = 0.01$, $\epsilon = 74$	$\alpha = 0.01$, $\epsilon = 73$	$\alpha = 0.01$, $\epsilon = 73$	$\alpha = 0.02$, $\epsilon = 55$	$\alpha = 0.02$, $\epsilon = 55$
$\alpha = 0.02$, $\epsilon = 55$	$\alpha = 0.01$, $\epsilon = 74$	$\alpha = 0.01$, $\epsilon = 73$	$\alpha = 0.01$, $\epsilon = 73$	$\alpha = 0.02$, $\epsilon = 55$	$\alpha = 0.02$, $\epsilon = 52$
$\alpha = 0.005$, $\epsilon = 85$	$\alpha = 0.01$, $\epsilon = 74$	$\alpha = 0.01$, $\epsilon = 73$	$\alpha = 0.01$, $\epsilon = 73$	$\alpha = 0.01$, $\epsilon = 73$	$\alpha = 0.02$, $\epsilon = 55$
$\alpha = 0.01$, $\epsilon = 75$	$\alpha = 0.02$, $\epsilon = 56$	$\alpha = 0.02$, $\epsilon = 55$	$\alpha = 0.02$, $\epsilon = 55$	$\alpha = 0.02$, $\epsilon = 55$	$\alpha = 0.01$, $\epsilon = 71$

compared to Linux's Ondemand governor and reducing power consumption by 9.45%, whereas, the agent achieved 6.32% reduction in peak temperature while improving performance by 7.85% compared to Linux's Performance governor and reducing power consumption by 8.45%. In the aforementioned experiments, the performance deadline for P-EdgeCoolingMode has not been set, yet the agent was able to improve performance at the same time reducing the thermal gradient. On the other hand, the Deep Q-Learning Dynamic Management is only able to reduce the operating temperature to 93.56°C (average) with a power consumption of 10.81 W while maintaining similar performance as executing Streamcluster on Linux's Ondemand governor, and hence, is capable of reducing the operating temperature by 1.52% (average) and reducing the power consumption by 1.82% (average) compared to peak temperature and power consumption. Therefore, the P-EdgeCoolingMode outperforms the Deep Q-Learning Dynamic Management while improving performance of the executing application.

We executed Whetstones benchmark for five times and here, we present the average result of all five executions. For Whetstones the total execution time, when run with the Linux's stock governor ondemand, is 121.56 secs with 206.8 times CPU throttling and an average temperature of 93.95° centigrade. With our approach, we achieved 4.11% reduction in temperature with a performance boost of 1.65% and no CPU throttling. We have also noticed that while using linear regression of temperature vs frequency instead of quadratic equation, on an average the computation takes 193 ms whereas the quadratic one takes 237 ms and hence has a speedup of 1.295x with no loss in accuracy over time.

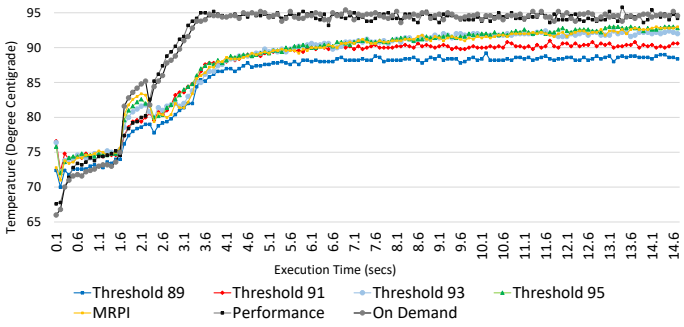

Fig. 5: Execution Time/Temperature Relationship For Streamcluster

4.3.2 Experiments on Kirin 659: On Kirin 659 MPSoC the task set for different actions (see Sec. 4.1.4) were executed on Linux's Interactive governor first to evaluate the performance of the default governor provided in the device. In Fig. 8, the average FPS of each action is shown. In the same figure (Fig. 8) you would also notice the average FPS of using P-EdgeCoolingMode

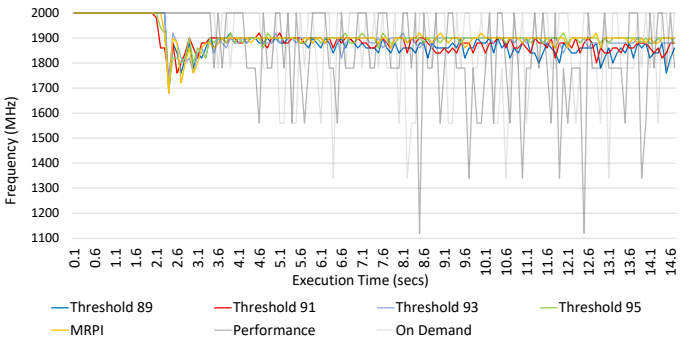

Fig. 6: Execution Time/Frequency Relationship For Streamcluster

Model	Exec Time (sec)	Avg. Temp. (°C)	Avg. Power (W)	Times Throttling
Th. 89	398.98	88.5	9.97	0
Th. 90	395.56	89.26	10.07	0.2
Th. 91	399.57	90.37	10.23	0.6
Th. 92	398.6	91.25	10.39	1
Th. 93	395.88	92.19	10.5	3.5
Th. 94	394.42	92.91	10.64	8.2
Th. 95	400.93	93.51	10.76	55.8
MRPI	406.63	93.7	10.79	119.2
Performance	433.01	94.08	10.89	649
On-Demand	429.78	94.24	11.01	630.2

Fig. 7: Result for different temperature threshold: Execution time, Avg. temperature, Avg. power, Avg. times throttling for different temperature threshold

(denoted as Avg. FPS (P-ECM)), which is actually the FPS performance deadline set for the chosen user action. The performance deadline for different actions in the P-EdgeCoolingMode were chosen after monitoring the Learning module of the agent and selecting the performance, which provided the satisfactory Quality of Service to users.

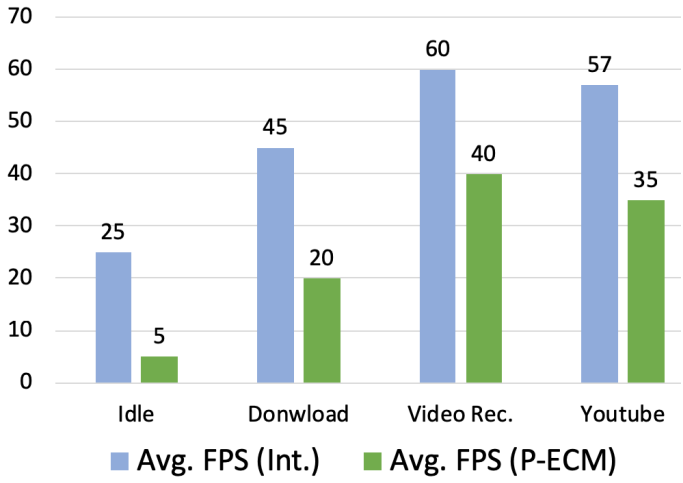

Fig. 8: FPS on Interactive Governor vs FPS on P-EdgeCoolingMode

In Fig. 9, we show the average power consumption in Watts for executing the user actions on Linux's Interactive governor and P-EdgeCoolingMode with performance deadline. In the figure, Avg. P. (Int.) denotes the average power consumption (Watts) for the chosen user action while using Interactive governor and Avg. P. (P-ECM) denotes the average power consumption (Watts) while using P-EdgeCoolingMode with performance deadline. From the figure, we could notice that using P-EdgeCoolingMode, we are able to achieve reduction in power consumption by 30.63% for Idle user action, 10.67% for downloading app action, 11.31% for taking picture/recording video and 13.25% for streaming Youtube on Chrome.

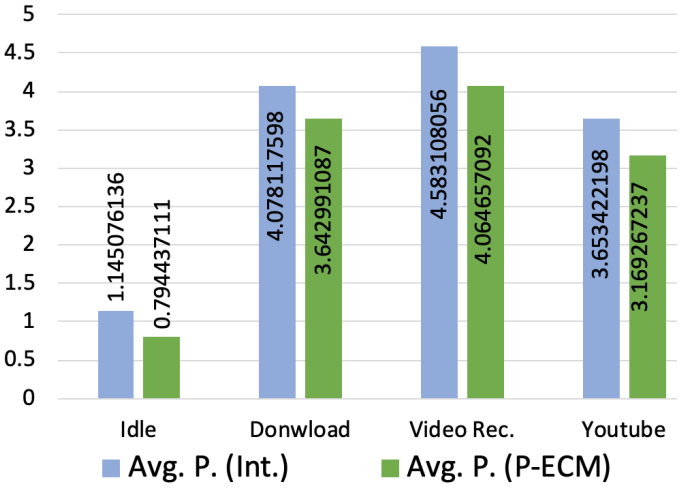

Fig. 9: Average power (Watt) consumption on Interactive governor vs P-EdgeCoolingMode

The Fig. 10, 11, 12 and 13 show individual power consumption for different user actions, where it could be noticed that using P-EdgeCoolingMode with performance deadline improves the power consumption while providing for performance requirement of that particular user action (chosen task set) in comparison to Linux's Interactive governor.

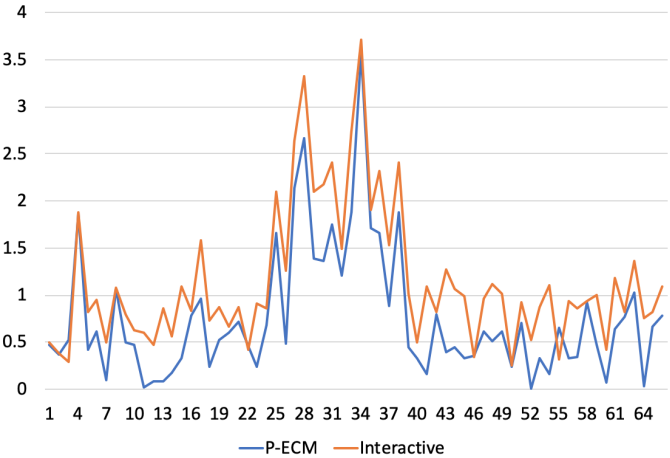

Fig. 10: Execution time (ms) vs Power consumption (Watt) during Idle action for Interactive governor (Interactive) and for P-EdgeCoolingMode (P-ECM)

5 Discussion

During our experiments, we noticed that power/temperature relationship was following a quadratic equation instead of exponential as is the well-known case. There could be several reasons for such behavior. One possibility is that we have run our experiments with the

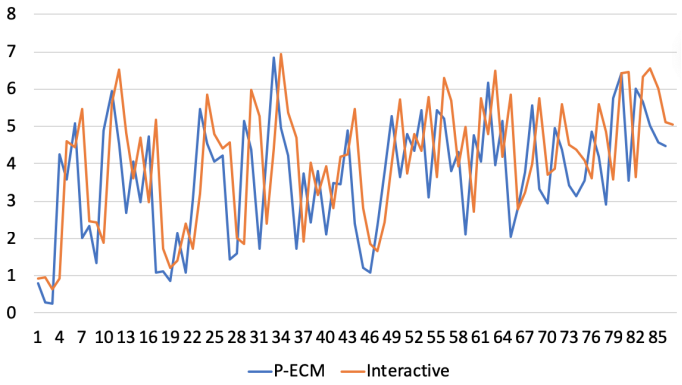

Fig. 11: Execution time (ms) vs Power consumption (Watt) during downloading an app from Google Play store while on Interactive governor (Interactive) and on P-EdgeCoolingMode (P-ECM)

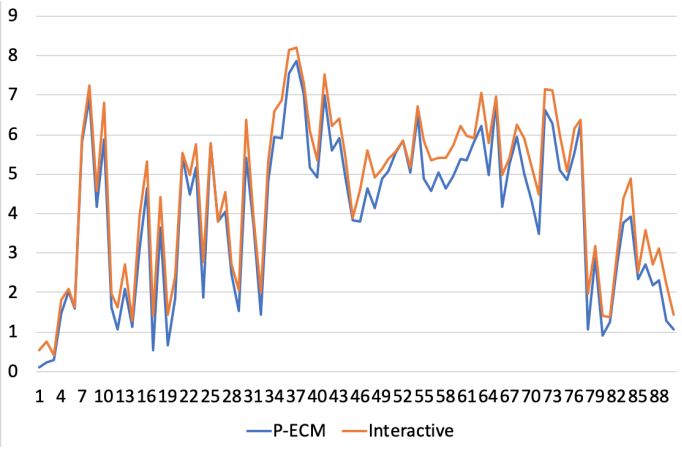

Fig. 12: Execution time (ms) vs Power consumption (Watt) during picture snapping/video recoding while on Interactive governor (Interactive) and on P-EdgeCoolingMode (P-ECM)

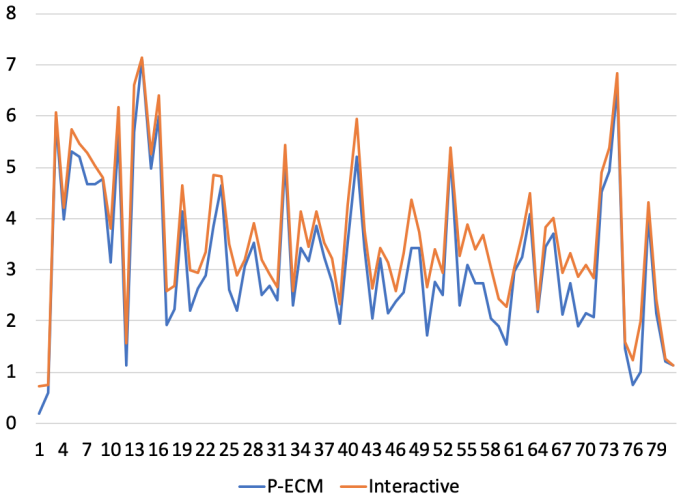

Fig. 13: Execution time (ms) vs Power consumption (Watt) during Youtube streaming on Chome while on Interactive governor (Interactive) and on P-EdgeCoolingMode (P-ECM)

active cooling mechanism (cooling fan with heatsink) on board of the Odroid XU4, which tried to physically regulate the temperature thus regulating the spatial thermal gradient as much as possible through active heat dissipation. Another point we have to keep in mind is that since our methodology acts as an intelligent agent to enhance the capacities of thermal regulation of the system instead of acting as a replacement for the already existing thermal management unit on the system, we did not turn off the TMUs for our experiments. We mapped the relationship between power and temperature after executing* our P-EdgeCoolingMode mechanism with a thermal threshold of 90° centigrades and the relationship follows a quadratic function (see Fig. 14). Since our proposed P-EdgeCoolingMode was acting as a thermal regulator, therefore, regulating the temporal thermal gradient and hence from the aforementioned figure (Fig. 14) we could see that the temperature is regulated (temperature variance stabilizes) and maintained below the thermal cap set by the OS.

We have also noticed that we achieve different results based on the time period of the day due to the difference in ambient temperature between night and day†. Fig. 7 shows the results of experiments performed during daytime when the ambient temperature was hottest. Thus if the same set of experiments are performed now the results might vary by a little percentage.

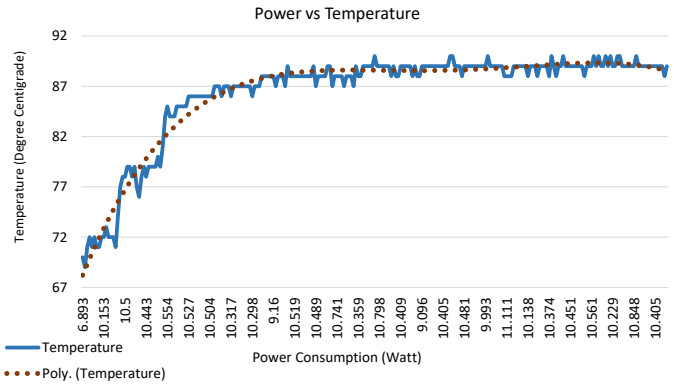

Fig. 14: Power/Temperature Relationship Using P-EdgeCoolingMode

6 Conclusion

In this paper, we have proposed a light-weight thermal management mechanism in the form of an intelligent agent, which is capable of monitoring and regulating the operating temperature of the CPU cores on MPSoCs, and pro-actively monitors performance at the same time. The efficacy of the methodology was evaluated by implementing and validating the mechanism on the Odroid-XU4 MPSoC and Huawei P20 Lite (Kirin 659 MPSoC), employing ARM's big.LITTLE architecture. The results of applying the agent-based thermal manager showed that, compared with the state-of-the-art power and temperature management approaches, the proposed approach was not only capable of reducing operating temperature of the CPU cores but at the same time improved performance of the executing application as well as reduction in power consumption.

*We ran Stremcluster benchmark from PARSEC and mapped the power consumption with the peak temperature of the big cluster.

†Ambient temperature between night and day varied by a couple of degrees, hence, affecting the results.

Acknowledgment

This work is supported by the UK Engineering and Physical Sciences Research Council EPSRC [EP/R02572X/1 and EP/P017487/1] and the authors would like to thank the people associated with National Centre for Nuclear Robotics (NCNR) and Extreme Environments for their support and feedback. Somdip would also like to thank everyone from the Embedded and Intelligent Systems Laboratory at the University of Essex for their feedback on this project.

7 References

1 'Exynos 5 octa (5422)'. (Samsung, . accessed: 2018-07-23. https://www.samsung.com/exynos
2 'Hisilicon kirin 650 (659)'. (Hisilicon, . accessed: 2018-07-23. http://www.hisilicon.com/en/Solutions/Kirin
3 Singh, A.K., Dziurzanski, P., Mendis, H.R., Indrusiak, L.S.: 'A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems', *ACM Computing Surveys (CSUR)*, 2017, **50**, (2), pp. 24
4 Singh, A.K., Leech, C., Reddy, B.K., Al.Hashimi, B.M., Merrett, G.V.: 'Learning-based run-time power and energy management of multi/many-core systems: current and future trends', *Journal of Low Power Electronics*, 2017, **13**, (3), pp. 310–325
5 Reddy, B.K., Merrett, G.V., Al.Hashimi, B.M., Singh, A.K. 'Online concurrent workload classification for multi-core energy management'. In: Design, Automation Test in Europe Conference Exhibition (DATE). (, 2018. pp. 621–624
6 Dey, S., Guajardo, E.Z., Basireddy, K.R., Wang, X., Singh, A.K., McDonald.Maier, K.D. 'Edgecoolingmode: An agent based thermal management mechanism for dvfs enabled heterogeneous mpsocs'. In: The 2019 32nd International Conference on VLSI Design (VLSID 2019) and 2019 18th International Conference on Embedded Systems. (, 2018.
7 DeVogeleer, K., Memmi, G., Jouvelot, P., Coelho, F. 'Modeling the temperature bias of power consumption for nanometer-scale cpus in application processors'. In: Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014 International Conference on. (IEEE, 2014. pp. 172–180
8 Aalsaud, A., Shafik, R., Rafiev, A., Xia, F., Yang, S., Yakovlev, A. 'Power–aware performance adaptation of concurrent applications in heterogeneous many-core systems'. In: Proceedings of the 2016 International Symposium on Low Power Electronics and Design. (ACM, 2016. pp. 368–373
9 Reddy, B.K., Singh, A., Biswas, D., Merrett, G., Al.Hashimi, B.: 'Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores', *IEEE Transactions on Multiscale Computing Systems*, 2017, pp. 1–14
10 Dey, S., Singh, A.K., McDonald.Maier, K.: 'Energy efficiency and reliability of computer vision applications on heterogeneous multi-processor systems-on-chips (mpsocs)', , ,
11 Chandramohan, K., O'Boyle, M.F. 'Partitioning data-parallel programs for heterogeneous mpsocs: time and energy design space exploration'. In: ACM SIGPLAN Notices. vol. 49. (ACM, 2014. pp. 73–82
12 Barik, R., Farooqui, N., Lewis, B.T., Hu, C., Shpeisman, T. 'A black-box approach to energy-aware scheduling on integrated cpu-gpu systems'. In: Proceedings of the 2016 International Symposium on Code Generation and Optimization. (ACM, 2016. pp. 70–81
13 Singh, A.K., Prakash, A., Basireddy, K.R., Merrett, G.V., Al.Hashimi, B.M.: 'Energy-efficient run-time mapping and thread partitioning of concurrent opencl applications on cpu-gpu mpsocs', *ACM Transactions on Embedded Computing Systems (TECS)*, 2017, **16**, (5s), pp. 147
14 Bienia, C. 'Benchmarking Modern Multiprocessors'. Princeton University, 2011
15 Chantem, T., Dick, R.P., Hu, X.S. 'Temperature-aware scheduling and assignment for hard real-time applications on mpsocs'. In: Proceedings of the conference on Design, automation and test in Europe. (ACM, 2008. pp. 288–293
16 Coskun, A.K., Rosing, T.S., Whisnant, K. 'Temperature aware task scheduling in mpsocs'. In: Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07. (IEEE, 2007. pp. 1–6
17 'Odroid-xu4'. (Hardkernel, . accessed: 2018-07-23. https://goo.gl/KmHZRG
18 'Huawei p20 lite'. (Huawei, . accessed: 2018-07-23. https://consumer.huawei.com/uk/phones/m/p20-lite/
19 Ghasemazar, M., Goudarzi, H., Pedram, M. 'Robust optimization of a chip multiprocessor's performance under power and thermal constraints'. In: Computer Design (ICCD), 2012 IEEE 30th International Conference on. (IEEE, 2012. pp. 108–114
20 Kamal, M., Iranfar, A., Afzali.Kusha, A., Pedram, M. 'A thermal stress-aware algorithm for power and temperature management of mpsocs'. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. (EDA Consortium, 2015. pp. 954–959
21 Carlson, T.E., Heirman, W., Eeckhout, L. 'Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations'. In: International Conference for High Performance Computing, Networking, Storage and Analysis (SC). (, 2011. pp. 52:1–52:12
22 Iranfar, A., Kamal, M., Afzali.Kusha, A., Pedram, M., Atienza, D.: 'Thespot: Thermal stress-aware power and temperature management for multiprocessor systems-on-chip', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018, **37**, (8)
23 Singla, G., Kaur, G., Unver, A.K., Ogras, U.Y. 'Predictive dynamic thermal and power management for heterogeneous mobile platforms'. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. (EDA Consortium, 2015. pp. 960–965
24 Bhat, G., Singla, G., Unver, A.K., Ogras, U.Y.: 'Algorithmic optimization of thermal and power management for heterogeneous mobile platforms', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2018, **26**, (3), pp. 544–557
25 Chetto, H., Silly, M., Bouchentouf, T.: 'Dynamic scheduling of real-time tasks under precedence constraints', *Real-Time Systems*, 1990, **2**, (3), pp. 181–194
26 Sprunt, B., Sha, L., Lehoczky, J. 'Scheduling sporadic and aperiodic events in a hard real-time system'. (CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1989.
27 Zhang, Q., Lin, M., Yang, L.T., Chen, Z., Khan, S.U., Li, P.: 'A double deep q-learning model for energy-efficient edge scheduling', *IEEE Transactions on Services Computing*, 2018,
28 ul Islam, F.M.M., Lin, M.: 'Hybrid dvfs scheduling for real-time systems based on reinforcement learning', *IEEE Systems Journal*, 2017, **11**, (2), pp. 931–940
29 Gupta, U., Mandal, S.K., Mao, M., Chakrabarti, C., Ogras, U.Y.: 'A deep q-learning approach for dynamic management of heterogeneous processors', *IEEE Computer Architecture Letters*, 2019, **18**, (1), pp. 14–17
30 Negnevitsky, M.: 'Artificial intelligence: a guide to intelligent systems'. (Pearson Education, 2005)
31 McDermott, D., Charniak, E.: 'Introduction to artificial intelligence', *Reading: Addison-Wesley*, 1985,
32 Russell, S.J., Norvig, P.: 'Artificial intelligence: a modern approach'. (Malaysia; Pearson Education Limited,, 2016)
33 Haugeland, J.: 'Artificial intelligence: The very idea. 1985', *Cited on*, 1985, p. 26
34 Rich, E., Knight, K.: 'Artificial intelligence', *McGraw-Hill, New*, 1991,
35 Dey, S., Kalliatakis, G., Saha, S., Singh, A.K., Ehsan, S., McDonald.Maier, K.: 'Mat-cnn-sopc: Motionless analysis of traffic using convolutional neural networks on system-on-a-programmable-chip', *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2018)*, 2018,
36 Thrun, S., Mitchell, T.M. 'Lifelong robot learning'. In: The biology and technology of intelligent autonomous agents. (Springer, 1995. pp. 165–196
37 Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka.Jr, E.R., Mitchell, T.M. 'Toward an architecture for never-ending language learning.'. In: AAAI. vol. 5. (Atlanta, 2010. p. 3
38 'Arm big.little technology'. (arm, . accessed: 2018-07-23. http://www.arm.com/
39 'Odroid smartpower2'. (Hardkernel, . accessed: 2018-07-23. https://www.hardkernel.com/main/products/prdt_info.php\?g_code=G148048570542
40 Curnow, H.J., Wichmann, B.A.: 'A synthetic benchmark', *The Computer Journal*, 1976, **19**, (1), pp. 43–49
41 'Implementing vsync'. (Android, . accessed: 2018-12-23. https://source.android.com/devices/graphics/implement-vsync