Improving Mobile Gaming Performance through Cooperative CPU-GPU Thermal Management

Alok Prakash[†], Hussam Amrouch[§], Muhammad Shafique[§], Tulika Mitra[†], Jörg Henkel[§]
[†]National University of Singapore, [§]Karlsruhe Institute of Technology,
Corresponding Author: tulika@comp.nus.edu.sg

ABSTRACT

State-of-the-art thermal management techniques independently throttle the frequencies of high-performance multi-core CPU and powerful graphics processing units (GPU) on heterogeneous multiprocessor system-on-chips deployed in latest mobile devices. For graphics-intensive gaming applications, this approach is inadequate because both the CPU and the GPU contribute towards the overall application performance (frames per second or FPS) as well as the on-chip temperature. The lack of coordination between CPU and GPU induces recurrent frequency throttling to maintain on-chip temperature below the permissible limit. This leads to significantly degraded application performance and large variation in temperature over time. We propose a control-theory based dynamic thermal management technique that cooperatively scales CPU and GPU frequencies to meet the thermal constraint while achieving high performance for mobile gaming. Experimental results with six popular Android games on a commercial mobile platform show an average 19% performance improvement and over 90% reduction in temperature variance compared to the original Linux approach.

1. INTRODUCTION

The continued success of Moore's Law has enabled integration of several high-performance compute cores into modern heterogeneous multiprocessor system-on-chips (MPSoCs) that drive the latest smart-phones, tablet PCs, hand-held gaming consoles, etc. For instance, the Samsung Exynos 5250 MPSoC, powering the popular Google Nexus 10 tablet, contains a multi-core ARM Cortex-A15 CPU alongside a high-performance ARM Mali T604 GPU. The presence of such high performance components in a portable device enables the execution of sophisticated applications such as video editing, immersive 3D games, etc. These applications require both the CPU and GPU to work in tandem in order to provide a smooth user experience. However, the simultaneous utilization of these processing units on a compact chip results in high temperature for both CPU and GPU. The CPU and GPU frequency have to be throttled to meet the thermal constraint albeit at the cost of reduced performance [21]. In current Android devices, the CPU frequency is controlled by the operating system, while the GPU frequency is separately adjusted by the corresponding device driver. As this approach ignores the performance coupling between CPU and GPU during thermal management, it leads to significant loss in performance and unsatisfactory user experience, especially for gaming workloads that stress both CPU and GPU for high performance. Moreover, the inefficient thermal management also results in large temporal variation in temperature, which in turn raises severe reliability concerns for such devices [10].

We propose a control-theoretic thermal management approach that *cooperatively* performs dynamic voltage-frequency scaling (DVFS) for CPU and GPU while running gaming workload. This synergistic management significantly reduces the temperature variance and improves the frames per second (FPS) performance metric. We focus on gaming applications because of their requirement to use both CPU and GPU during execution. Moreover, the user experience or performance of a gaming workload can be easily perceived as well as quantified in the form of FPS enabling easy evaluation of the proposed approach. Our choice of workloads and performance metric is not a limitation; our thermal management technique can also be adapted for other workloads and metrics. It should also be noted that while integrated CPU-GPU DVFS has been considered in the past for power management [7] [6], thermal management poses additional challenges, as discussed in the subsequent sections, that call for a new technique.

The novel contributions of this paper are:

1. We perform a detailed characterization of CPU and GPU temperature profile over various combinations of DVFS settings in a commercial MPSoC. This study enables us to identify the individual contribution of CPU and GPU in raising the overall MPSoC temperature.
2. Using the results from the characterization step, we derive models to predict the temperature of CPU, GPU as well as the overall MPSoC.
3. We propose a cooperative, control-theoretic dynamic thermal management technique that uses the thermal models to significantly reduce the temperature variance while achieving high application performance.
4. For realistic evaluations, we implement the proposed thermal management approach in the Linux kernel on a commercial Android platform and show its effectiveness with popular high-end mobile games.

2. RELATED WORK

The increasing demand for performance from current mobile devices have pushed designers to integrate multi-core CPUs along with high-performance GPUs within a single die. This, in turn, resulted in high power densities as well as elevated on-chip temperatures, thereby making thermal management essential in modern mobile MPSoCs in order to maintain safe temperature for users [25]. Additionally, maintaining the system thermal behavior at runtime within the safety limits, while achieving satisfactory user experience exacerbates the challenge.

DAC '16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2898031

In response to this challenge, DVFS has been effectively used to fulfill the required thermal constraints [24]. Operating systems used in mobile devices offer different DVFS governors for CPU, GPU, etc. that respond to various criteria (e.g., CPU utilization) [1]. However, these governors do not target thermal management. Instead, they rely on independent thermal management units (TMUs) for CPU and GPU that are triggered only after the temperature constraint is violated. These TMUs do not consider the performance coupling between CPU and GPU, thereby reducing application performance. In [6, 7], DVFS-based power management approach for such architectures have been proposed in order to provide users with flexible power-performance trade-offs, however they do not focus on thermal management. Similarly, several works [8, 11, 9] consider coordinated CPU-GPU power management but do not consider thermal issues.

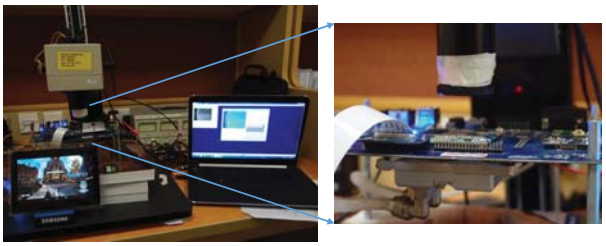
An application-driven thermal management policy that performs temperature-aware coding configuration selection along with frequency scaling has been proposed in [12] and targeted the high efficiency video coding. In [13], a QoS-aware DVFS technique for GPUs has been introduced to achieve an energy reduction with a higher QoS satisfaction rate than the on-demand DVFS policy. However, the work does not explore thermal implications of performing DVFS on GPUs embedded in mobile devices. The authors in [21] and more recently in [20] target dynamic thermal management on mobile devices while considering the temperature coupling between the processor and the battery. However, these works do not consider the interaction between the CPU and the integrated GPU that directly impacts the application performance, for on-chip thermal management.

Existing work has also targeted the thermal management issue for multi-core processors while accounting for inter-core thermal coupling, however, they do not focus on heterogeneous processing cores with producer-consumer relationship that also exhibit performance coupling [5]. Sharifi et al. in [23] propose a dynamic energy and thermal management techniques for heterogeneous embedded MPSoCs, however their algorithm does not consider the performance coupling between the various processing cores.

The authors in [16] discuss the power and thermal management for the AMD Trinity platform that consists of a CPU and GPU leading to performance coupling for certain applications. However, they do not control the GPU DVFS settings explicitly unlike the proposed algorithm in this paper. Additionally, they assume that the thermal coupling only affects the nearby cores when the temperature approaches the critical value. Authors in [14] target both CPU and GPU for thermal management while accounting for the ambient variations and user interactions. However, they assume that both the CPU cores and the GPU always consume equal power. This assumption does not hold true for gaming workloads as shown in [7] and necessitates more sophisticated distribution of the power/thermal budget for thermal management under gaming workloads. Recently, Singla et al. [15] has proposed a predictive dynamic thermal and power management for heterogeneous mobile platforms. They propose to turn off the big A15 cores in their platform one by one if a thermal violation is predicted. Little A7 cores and GPU cores are throttled subsequently, if further thermal violations are predicted. Their results show significant reduction in temperature variance when compared to Linux albeit at a loss in performance at similar temperature, especially for gaming workloads that require both CPU and GPU for high performance. In contrast, our proposed controller achieves higher performance than Linux while maintaining similar peak temperature and significantly reducing the temperature variance.

3. THERMAL CHARACTERIZATION

In this section we first describe our experimental setup followed by characterization of CPU-GPU thermal profile.



(a) Experimental Setup (b) Close-up of the lens and additional heatsink

Figure 1: Experimental setup

3.1 Experimental Setup

We use Arndale development platform [2] with Samsung Exynos 5250 MPSoC, which powers the latest Google Nexus tablet, for our experiments. The Exynos MPSoC contains a dual core ARM Cortex-A15 processor that can be clocked between 200 MHz to 1.7 GHz at an interval of 100 MHz. This provides us with 16 frequency settings for the CPU. The MPSoC also integrates a high performance Mali T604 quad-core GPU that is capable of 7 DVFS settings between 100 MHz to 533 MHz as shown in Table 1. The platform runs Android OS version 4.2.1 with kernel version 3.4.5. The typical Linux governor, "interactive", performs CPU DVFS based on its utilization, while the GPU driver does the same for the GPU.

Table 1: Available frequency settings for GPU

Frequency (MHz)	100	160	266	350	400	450	533

The Exynos 5250 MPSoC incorporates only one on-chip temperature sensor that is monitored by the thermal management unit (TMU). In order to analyze CPU and GPU temperature individually, we additionally employ a thermal camera that continuously captures the infrared images of the silicon die. The complete experimental setup along with the thermal camera is shown in Figure 1(a). Figure 1(b) shows the close-up view of the development board under the thermal camera. It can be observed that a heat sink is attached to the bottom of the device for heat dissipation. This is done due to the removal of the heat-spreader packaging from the Exynos MPSoC in order for the thermal camera to focus on the silicon die. The implementation details of the built thermal setup can be found in [4]. We use a peltier device similar to [12, 4] to appropriately control the temperature of this heat sink to ensure that the behavior of the modified chip remain close to the original chip. In the future, we expect the chip manufacturers to integrate more on-chip sensors in mobile MPSoCs for comprehensive thermal management. Indeed, the XU-3 Odroid platform that contains a newer high performance Exynos 5422 MPSoC incorporates temperature sensors for both CPU and GPU. As the ambient temperature plays an important role in any thermal analysis, special care is also taken to ensure the room temperature to always be approximately at 25 $\pm$ 0.2°C during experimentation.

3.2 Analyzing the Thermal Profiles

Figure 2 shows the effect of thermal throttling on CPU-GPU frequency and resulting performance for our experimental platform while running *Anomaly2* gaming application on Android. The top half of Figure 2 presents the frequency and temperature of CPU, GPU obtained from the thermal camera along with the FPS. The bottom half of Figure 2 shows a closeup view during thermal throttling and includes the GPU utilization curve that was excluded in

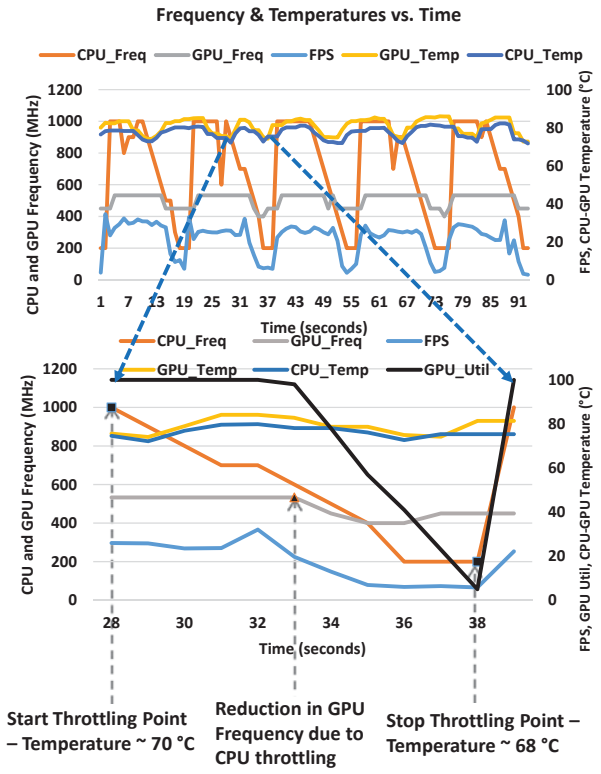
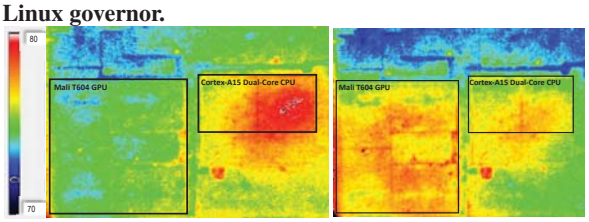


Figure 2: Effect of Thermal Throttling on frequency and FPS in Linux governor.

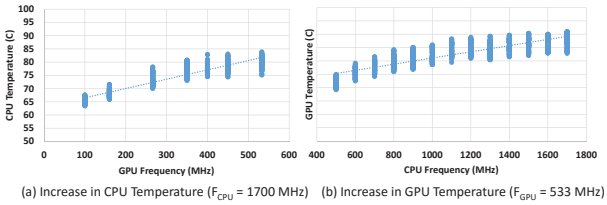


(a) CPU is hotter than GPU (b) GPU is hotter than CPU

Figure 3: On-chip temperature map showing variation in CPU-GPU temperature at different phases of gaming application.

the top figure for clarity. Every time the temperature hits the predefined threshold of 70°C, the Linux TMU throttles the CPU frequency and the performance (FPS) degrades. The GPU frequency is reduced by the device driver a little later; this reduction is not coordinated with the CPU frequency adjustment. The net effect is that the temperature continues to rise to around 79°C even after throttling CPU frequency due to thermal inertia [19]. Unlike power consumption, the temperature of a device is not only influenced by its current frequency, but also its past frequency values due to this phenomenon of thermal inertia. The chip only cools down when both CPU and GPU frequency have been throttled. Once the on-chip temperature reaches below the lower threshold of 68°C, the CPU frequency is increased immediately, producing a rise in FPS. The sudden changes or jitter in FPS is highly visible and creates an uneven user experience. Also, the temperature continuously oscillates between 68°C and 79°C and may create reliability issues.

Figure 3 shows infrared images, as captured by the thermal camera, of the MPSoC at two instances during the execution of a 3D gaming application. The figures clearly show that both the CPU and GPU get hot when stressed. However, the CPU and the GPU may get heated up at different points in time. For example, when the CPU is creating a scene or performing artificial intelligence computation, the CPU is hot while the GPU is cold. In contrast, when a scene is being rendered, the GPU is much more hot than the CPU.



(a) Increase in CPU Temperature (F_{CPU} = 1700 MHz) (b) Increase in GPU Temperature (F_{GPU} = 533 MHz)

Figure 4: CPU (or GPU) temperature rise with increasing GPU (or CPU) frequency while CPU (or GPU) frequency is kept constant.

Additionally, unlike power consumption, due to the close proximity of CPU and GPU, even a moderate increase in the temperature of one directly influences the temperature rise in the other. This phenomenon is known as *thermal coupling* and makes it challenging to predict the thermal behavior of the individual processing elements [16]. In order to gain further insights into the correlation between CPU-GPU frequencies and the resulting MPSoC temperature, we observe temperature profile at all possible frequency combinations of CPU and GPU. The CPU frequency is set between 500 MHz to 1700 MHz[1] and GPU frequency between 100 MHz to 533 MHz. Therefore, we have $(13 * 7 = 91)$ frequency combinations for the characterization step. We use *Epic Citadel* gaming application for this characterization because it can create reproducible workload during each run and is powered by the popular gaming engine, "Unreal Engine 3" [3] that stresses both CPU and GPU. At each CPU-GPU frequency setting, we cool down the chip to a fixed temperature for consistency in the experiments. We record the on-chip temperature sensor and the thermal camera values averaged across the entire benchmark run. We modify and recompile the original Android kernel to disable CPU frequency throttling until on-chip temperature reaches 100°C, thereby enabling us to clearly observe the correlation between CPU-GPU frequency and temperature without the interference of TMU.

Figure 4 shows the observed reading for the CPU and GPU from the thermal camera for 91 CPU-GPU frequency combinations. Figure 4(a) shows the change in CPU temperature during the application execution when CPU frequency is kept constant at 1700 MHz while the frequency of the GPU is varied from 100 MHz to 533 MHz. Similarly, Figure 4(b) shows the change in GPU temperature when its frequency is kept constant at 533 MHz while the CPU frequency is changed. It can be seen from these figures that due to the close proximity of the CPU and GPU in this MPSoC, the variation in the frequency and thus temperature, of one processing element directly and significantly influences the temperature of the other component. This confirms our motivation that in a mobile MPSoC, that closely packs such high performance components, we must perform cooperative thermal management for these components.

In summary, two key observations can be made here:

- The MPSoC temperature continues to increase past the threshold due to the uncoordinated thermal management of CPU and GPU as well as thermal inertia. This eventually leads to a significant throttling of both the CPU and GPU frequencies, as shown in Figure 2, in order to reduce the chip temperature. This, in turn, results in high variance in on-chip temperature, which cycles between approximately 68 to 79°C in a short span of time and may lead to reliability issues [10]. Additionally, the lack of coordination between CPU

[1]Frequency setting below 500 MHz is not used as it is too low for the demanding gaming applications to execute correctly and hangs the board.

and GPU throttling significantly degrades application performance. Existing work has shown that applications, such as games, that employ both CPU and GPU concurrently must manage their frequencies in a coordinated manner in order to achieve high performance [6, 7].

- Secondly, the thermal coupling between the CPU and GPU due to their close proximity in a mobile MPSoC, also necessitates cooperative thermal management because even a moderate increase of temperature in one has direct and significant influence on the temperature of the other processing element.

4. TEMPERATURE MODEL FOR CPU AND GPU

In order to propose a dynamic thermal management (DTM) algorithm, we develop models to estimate the temperature of each processing element while considering their power dissipation.

The estimation model is generated based on the results from the characterization step discussed in the previous section. During the characterization step, a benchmark application is executed on the platform running at various frequency combinations of CPU and GPU, while logging various metrics such as the CPU-GPU frequencies, their utilization, temperature, etc. It should be noted that the temperature is observed and logged from two different sources, namely, the single on-chip temperature sensor and externally by using the infrared thermal camera as shown in Figure 3. We use the maximum temperature values for the CPU and GPU regions for the purpose of thermal analysis.

Both dynamic and static power dissipations contribute to the thermal phenomenon in a system. However, it has been shown earlier that static power consumption can be modeled as a constant term while also adjusting the coefficients of the dynamic power component, without significant loss in accuracy [5]. Moreover, similar to [5], our model is based on the experimental data obtained directly from the platform while executing the test application. This further ensures the capturing of thermal contribution of the static power dissipation. Hence, considering the cubic-frequency relationship between power and frequency [22], the temperature of the CPU and GPU at a fixed frequency can be estimated with the following expressions:

$$T_c = \alpha_c * U_c * F_c^3 + \delta_c \tag{1}$$

$$T_g = \alpha_g * U_g * F_g^3 + \delta_g \tag{2}$$

$$T_s = \beta_c * T_c + \beta_g * T_g \tag{3}$$

where, T_c, U_c, F_c, T_g, U_g and F_g denote the estimated temperature, current utilization and frequency values of the CPU and GPU respectively, whereas $\alpha_c, \delta_c, \beta_c, \alpha_g, \delta_g$ and β_g denote the platform-specific constants. The utilization values are used to approximate the activity factor during the calculation of the dynamic power consumption. Equation 1 and 2 refer to the contribution of power dissipation towards the temperature of the individual processing elements. Next, the individual contribution of the processing elements towards the overall MPSoC temperature can be modeled by using equation 3. The constants for equation 1, 2 and 3 are calculated by performing curve fitting in MATLAB on the data obtained from multiple executions of 4 popular Android games. The average error for this training set of games as well as the 6 other games used for the experimental evaluation later in section 6 is 1.5%.

Using equations 1 and 2 at two frequency values F and F', temperature T' of a component can also be estimated at a different frequency F' and utilization U' using the following equations:

$$T'_c = T_c + \sigma_c * (U'_c * F_c'^3 - U_c * F_c^3) \tag{4}$$

$$T'_g = T_g + \sigma_g * (U'_g * F_g'^3 - U_g * F_g^3) \tag{5}$$

The constant terms for the temperature models described in equations 4 and 5 were also obtained from a regression analysis on the data from the characterization step in section 3.2 using MATLAB and achieved an average error of less than 4.47% and 3% for CPU and GPU respectively across all applications. The next section uses these equations to propose a control-theory based cooperative dynamic thermal management algorithm that controls the CPU and GPU DVFS settings in order to maintain the temperature of each processing element within a target threshold value.

5. DYNAMIC THERMAL MANAGEMENT ALGORITHM

The proposed control-theory based DTM algorithm is shown in Figure 5. It strives to keep the MPSoC temperature at or close to a pre-defined threshold by estimating and allocating the available thermal headroom individually to both CPU and GPU based on their utilization values. Due to the absence of individual sensors, the algorithm starts by estimating the CPU and GPU temperatures, $T^c_{est}(t)$ and $T^g_{est}(t)$, at time step (t) from the current temperature measured by the on-chip sensor $T^s_{meas}(t)$ and the reference CPU and GPU temperature $T^c_{ref}(t-1)$ and $T^g_{ref}(t-1)$, set in the previous iteration. To this end, we first calculate the estimated on-chip temperature $T^s_{est}(t)$ using $T^c_{ref}(t-1)$ and $T^g_{ref}(t-1)$ in equation 3, as follows:

$$T^s_{est}(t) = \beta_c * T^c_{ref}(t-1) + \beta_g * T^g_{ref}(t-1) \tag{6}$$

$T^c_{est}(t)$ and $T^g_{est}(t)$ can be calculated as:

$$T^c_{est}(t) = T^c_{ref}(t-1) * (T^s_{meas}(t)/T^s_{est}(t)) \quad \&$$
$$T^g_{est}(t) = T^g_{ref}(t-1) * (T^s_{meas}(t)/T^s_{est}(t)) \tag{7}$$

We also measure the current CPU and GPU utilization as well as frequency, $U^c_{meas}(t), F^c_{meas}(t)$ and $U^g_{meas}(t), F^g_{meas}(t)$ respectively. These measured values are obtained at the end of the previous iteration. Next, we calculate the target frequency settings, $F^c_{target}(t+1)$ and $F^g_{target}(t+1)$, for time step $(t+1)$, for both CPU and GPU in order to maintain a pre-defined utilization target. Hence, assuming a linear relationship between frequency and utilization, $F^c_{target}(t+1)$ and $F^g_{target}(t+1)$ can be calculated as:

$$F^c_{target}(t+1) = F^c_{meas}(t) * (U^c_{meas}(t)/U^c_{target}) \quad \&$$
$$F^g_{target}(t+1) = F^g_{meas}(t) * (U^g_{meas}(t)/U^g_{target}), \tag{8}$$

where U^c_{target} and U^g_{target} are pre-defined utilization target values to avoid either CPU or GPU to become a bottleneck or run at an unnecessarily high frequency [6]. Next, we use the current estimated temperature, $T^c_{est}(t)$ and $T^g_{est}(t)$, and equations 4 and 5, to predict the temperatures, $T^c_{pred}(t+1)$ and $T^g_{pred}(t+1)$, at the target frequency settings as follows:

$$T^c_{pred}(t+1) = T^c_{est}(t) + \sigma_c * (U^c_{target} * (F^c_{target}(t+1))^3 - U^c_{meas}(t) * (F^c_{meas}(t))^3) \quad \&$$
$$T^g_{pred}(t+1) = T^g_{est}(t) + \sigma_g * (U^g_{target} * (F^g_{target}(t+1))^3 - U^g_{meas}(t) * (F^g_{meas}(t))^3) \tag{9}$$

The predicted temperature values are used in the "Temperature Allocator" block to find the reference temperatures, $T^c_{ref}(t+1)$ and $T^g_{ref}(t+1)$, for CPU and GPU based on their utilization values while ensuring that the overall MPSoC temperature stays at or close to the pre-defined threshold value, T^s_{thres}. In the proposed controller, the reference temperature for the CPU is identified first as it provides much finer (13 DVFS settings) level of frequency and hence, temperature control than the GPU (only 7 DVFS settings).

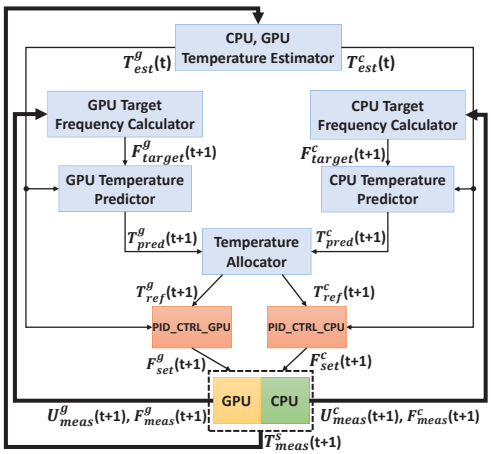


Figure 5: Cooperative Control-theory based DTM.

Hence, the reference temperature of the CPU is calculated as follows:

$$T^c_{ref}(t+1) = MIN(T^c_{pred}(t+1), T^c_{thres}) \quad (10)$$

where T^c_{thres} is the threshold temperature for the CPU.

The reference temperature for the GPU depends on the selected reference temperature for the CPU in order to maintain the overall chip temperature at the desired value. Hence, using equation 3, we first calculate the maximum permissible GPU temperature given the CPU temperature, as follows:

$$T^g_{max}(t+1) = (1/\beta_g)(T^s_{thres} - \beta_c * T^c_{ref}(t+1)) \quad (11)$$

Next, the reference temperature for the GPU is calculated as:

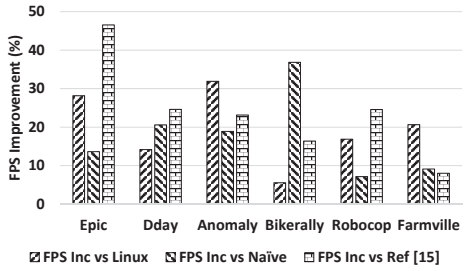
$$T^g_{ref}(t+1) = MIN(T^g_{pred}(t+1), T^g_{max}(t+1)) \quad (12)$$

In order to ensure the stability of the PID controller, the reference temperature values are chosen close to individual pre-defined threshold values of CPU and GPU temperature, while satisfying equation 3. These reference temperature values are then fed into the PID controller along with the current estimated temperatures of CPU and GPU, $T^c_{est}(t)$ and $T^g_{est}(t)$, to find the suitable frequencies for the CPU and GPU, $F^c_{set}(t+1)$ and $F^g_{set}(t+1)$, for the time step $(t+1)$. The algorithm sets the CPU and GPU frequencies and similar to [15], waits for 1 second before measuring the frequency, utilization and temperature values for the next iteration. A cycle period of 1 sec is chosen empirically because the chip temperature rises or falls slowly after applying the necessary frequency settings. The threshold values for temperature (74°C, 75°C for CPU and the chip respectively) and the target utilization (70% and 85% for CPU and GPU respectively) are also obtained empirically from extensive experimentation on the platform to ensure that the MPSoC does not violate the thermal constraints, while still achieving high performance. The proposed DTM algorithm is implemented in our target platform for verification and the results obtained are discussed in the next section.

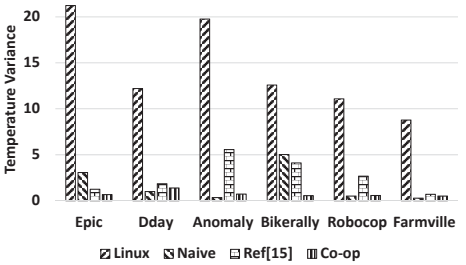
6. EXPERIMENTS AND RESULTS

To showcase the effectiveness of the proposed coordinated control-theory based thermal management technique, we compare it against the following state-of-the-art solutions.

- Linux: The original Linux TMU on this platform (Refer Section 3.2).
- Naive: An un-coordinated control theory based approach that does not consider the contribution of CPU and GPU on either the resulting chip temperature or the performance. The individual PID controllers, one each for CPU and GPU, simply strive to maintain the CPU and GPU temperature independently at pre-defined threshold values. The results from this controller serve as a baseline for comparison with the proposed co-operative thermal management solution that also accounts for CPU and GPU dependencies on the contradictory requirements of both performance (FPS) as well as thermal management.
- Ref [15]: A state-of-the-art technique proposed recently that does not take into account the dependency of CPU and GPU on the resulting application performance. This controller has been detailed in Section 2.
- Co-op: The proposed technique presented in this paper.



(a) Performance improvement of cooperative solution versus other approaches



(b) Temperature Variance Reduction

Figure 6: Improvement in FPS and Temperature Variance.

We choose 6 popular games from the Android Play Store to test the proposed techniques. These games are different from the ones used to create the models for Equations 1 to 5.

Figure 6(a) shows the increase in FPS, the performance metric used for gaming workloads, for the various games by using the proposed controller over Linux, Naive and the Ref [15] solutions respectively. It is noteworthy that the proposed controller achieves an average of over 19%, 17% and 23% improvement in FPS when compared to the Linux, Naive and the Ref [15] solutions respectively. While the authors in [15] also targeted both CPU and GPU for thermal management, they lacked the coordination between the two processing elements that is essential to achieve better performance for applications, such as games that employ both CPU and GPU. Instead, they proposed to perform DVFS for CPU and GPU sequentially, starting from the high performance Cortex A15 CPU and only reducing the GPU frequency as a last resort. In contrast, the proposed cooperative approach in this work manages the DVFS settings for both CPU and GPU based on their utilization as well as the thermal contribution at every step. This strategy not only helps in achieving higher performance, but also stabilizes the chip temperature as confirmed by Figure 6(b) that shows the statistical variance in the temperature samples over time.

In addition, as can be observed from Columns 2 through 5 of Table 2, the proposed approach also manages to achieve similar average temperature for all the games as the Naive and Ref [15] controllers. It is also worth noting that the applications show two distinct behaviors. The first four games in the table, exhibit lower

Table 2: Average And Max Temperature Comparison

	Average Temperature				Max Temperature			
Apps	Linux	Naive	Ref[15]	Co-op	Linux	Naive	Ref[15]	Co-op
Epic	77	75	76	75	83	79	79	76
Dday	75	75	76	75	79	80	79	77
Anomaly	78	75	76	75	83	77	79	77
Bikerally	76	75	76	75	80	80	79	77
Robocop	74	76	76	75	78	78	79	77
Farmville	74	75	75	75	77	77	76	76

or similar average temperature as Linux while using the proposed controller. These games are highly graphics intensive and hence they run at a very high temperature both in Linux as well as the proposed controller. However, the key difference is that, while the temperature in Linux varies rapidly by more than 10°C, the proposed controller maintains it tightly around the defined threshold and hence reduces the temperature variance over time as confirmed from Figure 6.(b). In fact, it can be seen that even the Naive and Ref [15] controllers reduce the variance in temperature to a large extent for all applications, however the proposed cooperative controller consistently performs similar or better than the existing solutions.

On the other hand, the applications *Robocop* and *Farmville*, are not as intensive and hence run at a lower temperature in the Linux controller. The proposed controller still strives to maintain the MPSoC temperature at its pre-defined threshold, thereby resulting in a slight increase in average temperature when compared to Linux. It should be noted that the increase in temperature for these two applications is just over 1-2% in the worst case. Moreover, the average temperature is still below the threshold and the proposed controller successfully mitigates the rapid variation in temperature that can raise reliability concerns. Lastly, it can be observed from columns 6 through 9 in Table 2 that unlike the Linux, Naive and the Ref [15] solutions, the cooperative thermal management strategy also does not allow the chip to heat up significantly at any point during the gameplay, as the overall chip temperature is taken into account while selecting the DVFS settings for the CPU and GPU at each control step. These results clearly shows the strength of the proposed coordinated approach for CPU-GPU thermal management for mobile games or even other applications that stress both the CPU and GPU during execution.

7. CONCLUSION

In this paper, we performed a detailed characterization of the thermal behaviour of the CPU and GPU in a commercial mobile MPSoC and showed that both CPU and GPU contribute towards the rise in chip temperature and hence must be managed cooperatively to achieve high performance with robust thermal management. We also derived models to estimate the temperature of CPU and GPU as well as the overall MPSoC with an estimation error of less than 5%. Next, we proposed a control-theoretic approach for cooperative CPU-GPU thermal management and verified it on a commercial mobile MPSoC. Our approach reduced the variance in on-chip temperature by more than 90% along with a reduction in the maximum temperature. At the same time, the average FPS performance increased by 19%, 17% and 23% when compared to the existing Linux-, an un-coordinated control-theory based Naive- and a recently proposed (Ref [15])- technique, respectively.

8. ACKNOWLEDGMENTS

The authors would like to thank Dr Thannirmalai Muthukaruppan for his invaluable support during the initial set up of the experimental platform together with the team at KIT, Germany. This work was partially supported by the Singapore Ministry of Education Academic Research Fund Tier 2 MOE2012-T2-1-115, the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre Invasive Computing [18](SFB/TR 89, B3- http://invasic.de) and the priority program Dependable Embedded Systems [17](SPP 1500, VirTherm3D- http://spp1500.itec.kit.edu).

9. REFERENCES

[1] Android Governors. http://goo.gl/8j1Eqo.
[2] Arndale board 5250. http://goo.gl/1ZCSNX.
[3] Epic Citadel, Unreal Engine. http://goo.gl/xdsQMr.
[4] H. Amrouch and J. Henkel. Lucid infrared thermography of thermally-constrained processors. In *ISLPED*, 2015.
[5] A. Bartolini et al. Thermal and Energy Management of High-Performance Multicores: Distributed and Self-Calibrating Model-Predictive Controller. *Parallel and Distributed Systems, IEEE Trans.*, 2013.
[6] A. Pathania et al. Integrated CPU-GPU power management for 3D mobile games. In *DAC*, 2014.
[7] A. Pathania et al. Power-performance modelling of mobile gaming workloads on heterogeneous mpsocs. In *DAC*, 2015.
[8] C. Wei-Ming et al. A User-Centric CPU-GPU Governing Framework for 3D Games on Mobile Devices. In *ICCAD*, 2015.
[9] C. Y. Hsieh et al. Memory-aware cooperative CPU-GPU DVFS governor for mobile games. In *ESTIMedia*, 2015.
[10] D. Brooks et al. Power, thermal, and reliability modeling in nanometer-scale microprocessors. *IEEE Micro*, 2007.
[11] D. Kadjo et al. A Control-theoretic Approach for Energy Efficient CPU-GPU Subsystem in Mobile Platforms. In *DAC*, 2015.
[12] D. Palomino et al. hevcDTM: Application-driven dynamic thermal management for high efficiency video coding. In *DATE*, 2014.
[13] D. You et al. Quality of service-aware dynamic voltage and frequency scaling for embedded GPUs. *Computer Architecture Letters*, 2014.
[14] F. Paterna et al. Modeling and Mitigation of Extra-SoC Thermal Coupling Effects and Heat Transfer Variations in Mobile Devices. In *ICCAD*, 2015.
[15] G. Singla et al. Predictive dynamic thermal and power management for heterogeneous mobile platforms. In *DATE*, 2015.
[16] I. Paul et al. Cooperative boosting: Needy versus greedy power management. In *ISCA*, 2013.
[17] J. Henkel et al. Design and architectures for dependable embedded systems. In *CODES+ISSS*, 2011.
[18] J. Henkel et al. Invasive manycore architectures. In *ASP-DAC*, 2012.
[19] M. Frankiewicz et al. Investigation of heat transfer in integrated circuits. *Metrology and Measurement Systems*, 2014.
[20] O. Sahin et al. On the impacts of greedy thermal management in mobile devices. *Embedded Systems Letters, IEEE*, 2015.
[21] Q. Xie et al. Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor. In *ICCAD*, 2013.
[22] R. Ahmed et al. Temperature minimization using power redistribution in embedded systems. In *VLSI Design*, 2014.
[23] S. Sharifi et al. Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs. In *ASP-DAC*, 2010.
[24] Y. Lei et al. Happe: Human and application-driven frequency scaling for processor power efficiency. *Mobile Computing, IEEE Trans.*, 2013.
[25] K. Sekar. Power and thermal challenges in mobile devices. In *MobiCom*, 2013.