Providing Sustainable Performance in Thermally Constrained Mobile Devices

Onur Sahin and Ayse K. Coskun
ECE Department, Boston University, Boston, MA, USA
{sahin, acoskun}@bu.edu

ABSTRACT

State-of-the-art smartphones can generate excessive amounts of heat during high computational activity or long durations of use. While throttling mechanisms ensure safe component and outer skin level temperatures, frequent throttling can largely degrade the user-perceived performance. This work explores the impact of multiple different thermal constraints in a real-life smartphone on user experience. In addition to high processor temperatures, which have traditionally been a major point of interest, we show that applications can also quickly elevate battery and device skin temperatures to critical levels. We introduce and evaluate various thermally-efficient runtime management techniques that slow down heating under performance guarantees so as to sustain a desirable performance for maximum durations. Our techniques achieve up to 8x longer sustainable QoS.

1. INTRODUCTION

Generational trends in mobile system-on-chip (SoC) designs have shown an increase in the peak power levels beyond thermal design limits due to integration of aggressive multicore CPUs (i.e., out-of-order and speculative multi-issue pipelines [17]) and high-end graphics accelerators (GPUs) [12]. Excessive power densities can cause maximum chip temperatures to quickly reach critical levels while running computationally demanding applications [8, 19, 21, 22].

While CPU temperature has been a major focus of research so far [10, 22], mobile devices are also constrained by the surface temperature limits. Unlike larger scale computing platforms (e.g., desktop computer, server) or mobile development boards (e.g., Odroid-XU3 [4]), commercial smartphones need to maintain touch surface (skin) temperatures within human comfort levels (e.g., 34 °C-43 °C [11]). The problem becomes even more apparent and crucial for mobile devices due to inherent limitations in cooling capabilities imposed by small form-factors and power restrictions.

Vendors incorporate thermal throttling policies to limit the maximum SoC temperature and maintain skin level temperatures within human comfort levels. Such policies adaptively reduce CPU power via dynamic voltage and frequency scaling (DVFS) or core offlining upon violating predefined thermal thresholds. Thermal throttling, however, can incur severe degradations in user quality-of-service (QoS)[1] and impair overall user experience due to CPU slowdown [7, 8].

Current runtime techniques in thermally constrained mobile platforms maximize performance under temperature limits in case of increased computation demand. While delivering short bursts of high QoS, such an approach accelerates heating and further magnifies the performance impacts of thermal throttling over extended durations of application use (e.g., as in gaming, streaming etc.) as shown by our prior work [20, 21]. Mobile users expect consistent and acceptable QoS while using applications over longer durations. Providing the users with *longer durations of sustainable QoS* requires thermally-efficient runtime strategies, as opposed to existing greedy approaches, to make more conservative usage of available thermal headroom for maximum durations.

In this paper, we demonstrate the performance impacts of the several different thermal constraints that exist in modern mobile platforms and propose a *thermally-efficient QoS management* approach to achieve longer durations of sustainable performance. We quantify the QoS loss incurred by thermal throttling for maintaining safe CPU and skin temperatures on smartphones using a set of real-life mobile applications. We present a number of novel observations and runtime techniques for both homogeneous and emerging heterogeneous mobile processors. Our techniques exploit thermal time constants via *fine-grained DVFS* and consider *CPU-GPU thermal couplings* and *thread criticality* in scheduling decisions to improve thermal efficiency. We perform all experiments on state-of-the-art mobile platforms with homogeneous and heterogeneous multicore CPUs. Our evaluations under both CPU and skin temperature constraints show up to 8x longer durations of sustainable QoS.

2. EXPERIMENTAL METHODOLOGY

Platform and Measurements: Our experiments are based on 3 mobile platforms described in Table 1. Qualcomm MDP8974 and Google Nexus 5 share the same chipset. Snapdragon 800 SoC in MDP8974 and Nexus 5 implements a homogeneous multi-core architecture with 4 Krait 400 CPU cores. We choose Nexus 5 as a representative commercial smartphone platform. The Exynos 5422 SoC in Odroid-XU3 mobile development platform integrates a big.LITTLE heterogeneous multi-core CPU with 4 high performance/power A15 cores in addition to 4 low-power A7 cores. We measure power and temperature using the built-in sensors. Odroid-XU3 allows measuring CPU, GPU and DRAM power consumptions individually while we can only measure the total

[1] QoS refers to a metric used to quantify the performance experienced by the user (e.g., frames-per-second (FPS) or response latency). While QoS could be application specific, our techniques are not specific to metric selection and can be applied broadly.

ESTIMedia'16, October 01-07, 2016, Pittsburgh, PA, USA

DOI: http://dx.doi.org/10.1145/2993452.2994309

platform power for Nexus 5 and MDP8974. FPS is measured by querying the logs generated by the *SurfaceFlinger* Android system service. We sample the number of executed instructions via Linux *perf_event* API. For consistent temperature measurements, before each experiment, we leave the platforms idle until CPU, skin and battery temperatures stabilize around initial idle temperature levels.

Table 1: Mobile devices used for experiments.

Device	Nexus 5 and Qualcomm MDP8974	Odroid-XU3
SoC	Qualcomm Snapdragon 800	Samsung Exynos 5422
CPU	Krait 400	ARM A15 + A7
Cores	4	4 + 4
CPU Freq.	2.2 GHz	2.1 GHz + 1.5 GHz
GPU	Adreno 330	Mali-T628
GPU Freq.	450 MHz	543 MHz

Applications: We experiment with a diverse set of benchmarks. We use two computing kernels, *FFT* and *SOR*, from the Scimark Java benchmark suite [18] and *H264* video encoding application from SPEC CPU2006 [13]. For these applications, throughput (instructions/sec) is measured as an indicator of QoS. We also use Heartbeats [14] instrumented version of the *bodytrack* computer vision application from the PARSEC suite [9]. Heartbeat framework allows to monitor application-specific QoS using a standardized interface and, for the *bodytrack* application, one heartbeat is emitted whenever the processing of one scene is completed. *Aquarium* [2], *Rain* [6] and *Pearl Boy* [5] are WebGL animations that we run within the Chrome web browser. *Edge of Tomorrow* and *Real Racing* are two representatives high-end gaming applications while *Mx Player* and *Rock Player* are common video player applications for Android. We use FPS as a measure of user experience for these applications.

Runtime Management: Google Nexus 5 smartphone employs a threshold based hysteresis skin temperature control policy, which adjusts the maximum CPU frequency limits. This policy reduces the maximum operating frequency from 2.2 GHz to 1.9 GHz, 1.5 GHz and 1.1 GHz when the skin temperature reaches 40 °C, 42 °C and 44 °C, respectively. Similarly, frequency limits are scaled up at the temperature levels of 38.5 °C, 40.5 °C and 42.5 °C. We implement a similar skin temperature control mechanism on MDP8974 as well. SoC temperature is controlled by a PID controller in all platforms. *Ondemand governor* in Nexus 5 and MDP8974 and *Interactive governor* in Odroid-XU3 are the default DVFS management mechanisms. Both governors adjust frequency according to CPU load [1]. In our policy implementations, we use *cpufreq* for DVFS management and Linux *sched_setaffinity* interface for enforcing thread mapping decisions. In Odroid-XU3, default HMP scheduler [3] determines big or LITTLE core execution for a task depending on its weighted average CPU load.

3. THERMAL CONSTRAINTS IN MODERN SMARTPHONES

Through experiments on real-life commercial smartphones and development platforms, this section illustrates and quantifies the potential loss in user experience due to multiple thermal constraints. We show how different applications can suffer from different constraints and illustrate the potential benefit of a platform-level thermal management approach.

Demonstrating throttling and performance unsustainability on real-life platforms. Figure 1 [20] shows the QoS degradation over time for various real-life applications (involving gaming, media streaming) running on a Odroid-XU3 platform. Over the 10 minutes of execution, CPU temperature induced throttling incurs significant QoS loss over time for all applications, reaching up to 50% degradation for *Aquarium*. Figure 2 shows the actions taken by the skin temperature control policy on a Nexus 5 smartphone during a 10 minute *Edge of Tomorrow* gameplay and its impact on QoS (i.e., FPS). As the skin temperature reaches 40 °C after 62 seconds, the policy starts to cap the maximum frequency of the CPU DVFS governors while also reducing the screen brightness to slowdown heating. At the end of 200 seconds of gameplay, the maximum frequency for the CPU governors is reduced from 2.2 GHz to 1.2 GHz by 55% while the display brightness is reduced from 100% to 70%. Due to reduced CPU frequency, the QoS degrades by as much as 40% (i.e., from 34 FPS to 20 FPS). This example signifies how the skin level thermal constraints (in addition to CPU temperature) limit the user experience in current generation smartphones under extended durations of use.

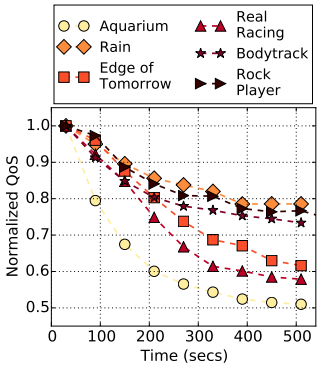


Figure 1: QoS degradation over time [20].

SoC vs skin temperature constraints. We demonstrate whether thermal throttling is triggered due to skin or chip thermal constraints depends on application's power profile. The average power dissipation for our applications are indicated in Figure 4a. Two applications with the highest power usage, *SOR* and *FFT* kernels, exhaust CPU thermal headroom before the skin temperature reaches to critical levels. This case is shown in Figure 3a for *FFT*. The CPU temperature reaches to 90 °C critical threshold in 6 seconds and QoS starts dropping due the reduced maximum frequency. It is around 29 seconds when the skin tempera-

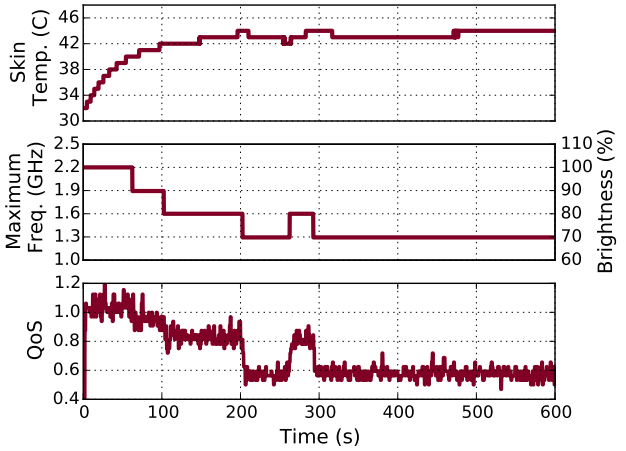
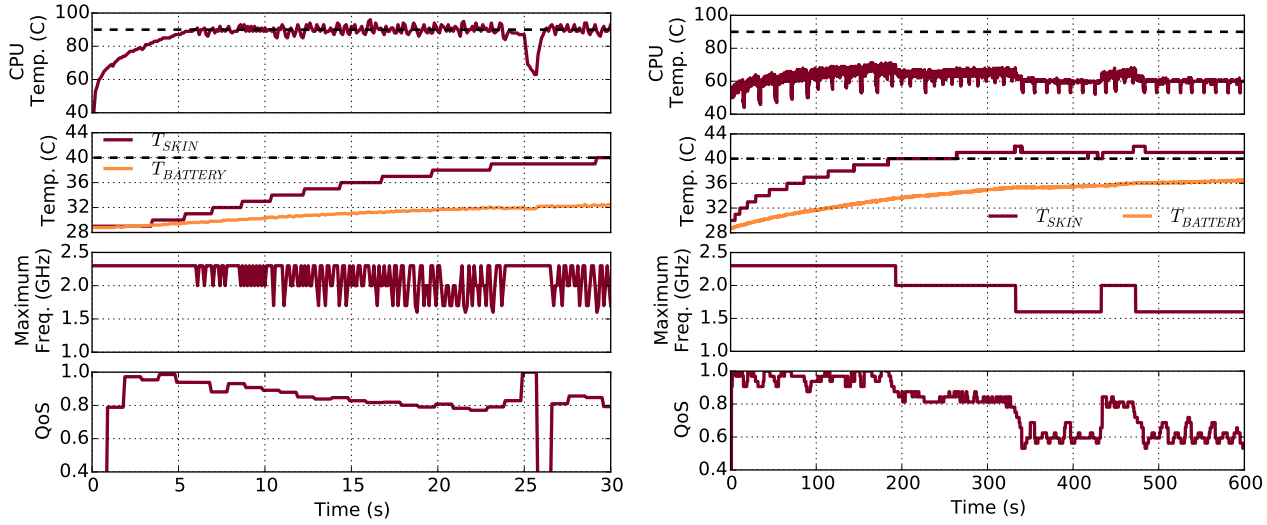


Figure 2: Thermal throttling policy in Nexus 5 reducing screen brightness and maximum CPU frequency due to elevated skin temperature while running *Edge of Tomorrow*.



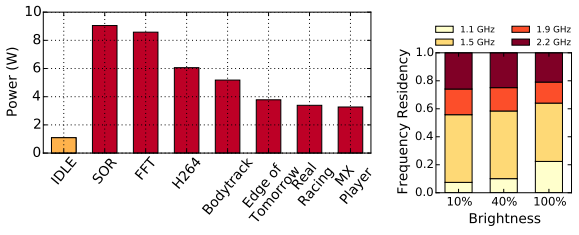
(a) FFT application.

(b) MX Player application.

Figure 3: Temperature, maximum frequency and QoS traces for *FFT* and *MX Player* applications. The maximum frequency levels are adaptively reduced when the CPU or the skin temperature temperatures reach critical thresholds (dashed lines).

ture reaches to 40 °C limit as well. For the *MX Player* video application in Figure 3b, on the other hand, throttling starts due to critical skin temperature level after 184 seconds.

For the cases where the frequency is scaled down aggressively due to skin temperature control, CPU thermal headroom could be inefficiently utilized. For instance, in Figure 3b, lowering the CPU frequency to reduce the skin temperature (despite the CPU not being thermally constrained) leads to as much as 30 °C waste in CPU thermal headroom for a 90 °C maximum threshold. Thus, if we could control the skin temperature with other knobs (than CPU DVFS), the CPU could have utilized higher frequencies due to the available thermal headroom. We demonstrate display brightness as an example of such control knob. Figure 4b illustrates the frequency residencies for a 10 minute *Edge of Tomorrow* gameplay under 3 brightness levels. Lowering the brightness slows down skin level heating and allows the CPU to utilize its available thermal headroom and operate at higher frequencies for a longer time. This motivates building *platform-level thermal management approaches* where the power states of major power consuming components (e.g., CPU, GPU, display etc.) are managed in coordination for maximizing user experience while reducing total power dissipation to reduce battery and skin temperatures.



(a) (b)

Figure 4: (a) Average power dissipation of Nexus 5 for different applications. "IDLE" indicate the power consumption when device is left idle. (b) Throttling and CPU frequency residencies under different screen brightness levels for the *Edge of Tomorrow* gaming application.

Table 2 summarizes the maximum temperatures, sustained QoS durations and QoS loss on Nexus 5. The maximum CPU temperature limit ($T_{CPU,lim}$=90 °C) is violated in the first 6 seconds for *FFT* and *SOR* while the other applications are primarily constrained by the skin temperature constraints ($T_{SKIN,lim}$=40 °C). All applications eventually reach the skin temperature limit. *MX Player* achieves the longest duration (184.1 sec) without throttling due to its lowest power consumption (Figure 4a). We observe 21.6% to 49% QoS loss for different applications due to thermal throttling. We measure battery temperatures as high as 40.8 °C. Such high battery temperatures further accelerates CPU and skin level heating due to thermal couplings between the battery and the other components [23].

Table 2: Summary of results on Nexus 5.

App.	T_{max} (CPU)	T_{max} (Skin)	T_{max} (Battery)	Time to $T_{CPU,lim}$	Time to $T_{SKIN,lim}$	QoS Loss
FFT	96 °C	48 °C	40.2 °C	5.9 sec	29.2 sec	48.1%
SOR	98 °C	49 °C	40.8 °C	4.5 sec	29.4 sec	49.0%
H264	88 °C	44 °C	36.1 °C	-	37.3 sec	48.1%
Bodytrack	85 °C	44 °C	37.7 °C	-	53.9 sec	44.9%
Aquarium	67 °C	44 °C	37.6 °C	-	138.6 sec	44.4%
Edge of T.	67 °C	44 °C	37.7 °C	-	122.1 sec	44.1%
Real Racing	65 °C	44 °C	37.4 °C	-	160.1 sec	21.6%
MX Player	71 °C	42 °C	36.5 °C	-	184.1 sec	38.7%

4. THERMALLY-EFFICIENT QOS MANAGEMENT

The goal behind our work is to enable *thermally-efficient QoS management* for providing the mobile system users with longer periods of acceptable performance. Rather than greedily maximizing performance under thermal constraints, in order to utilize the thermal headroom more efficiently, we propose to integrate user/application QoS requirements into thermal management [21]. We design practical runtime QoS management techniques to deliver desired user performance and develop several novel insights to improve the thermal-efficiency of QoS management. This section describes these insights and techniques. We plan to investigate, in our future work, the use of non-CPU control knobs to further reduce skin temperatures for the applications that are primarily constrained by skin temperature limits.

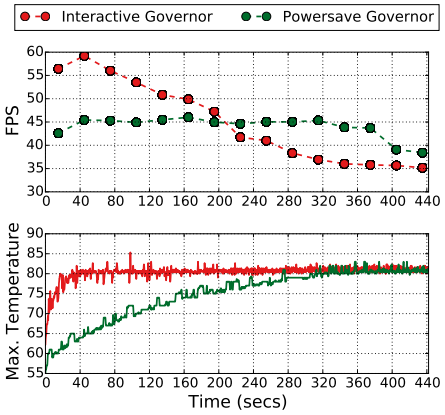


Figure 5: A case for QoS-temperature tradeoff for achieving longer sustainable QoS for a gaming application (*Real Racing*) running on a Odroid-XU3 mobile platform [20].

4.1 Managing QoS-Temperature Tradeoff

We show that users or system-level policies can proactively trade off QoS for extra thermal headroom for an application that is likely to suffer from thermal throttling. This extra thermal headroom allows the application to sustain the same performance level for longer durations. The idea of trading off the application QoS has been widely employed for energy savings [15]. Trading off the QoS for thermal headroom, however, is a novel insight brought by our work [21] to address the QoS unsustainability problem in mobile devices we have demonstrated in Section 3.

We demonstrate this intuition on a real hardware. Figure 5 [20] shows the QoS (frames-per-second (FPS)) and temperature traces for a gaming application we ran with two different DVFS governors [1] in Android. The default *Interactive* governor provides the highest QoS initially. However, it quickly (in 40 seconds) exhausts the SoC thermal headroom and QoS significantly drops over time due to thermal throttling. *Powersave* governor, which uses the lowest available frequency of big cores, sustains the QoS above 45 FPS for almost twice as long by mitigating the thermal throttling further to 380 seconds. Thus, providing techniques to exploit QoS-temperature tradeoff can allow users or system-level policies to extend the durations of sustainable QoS.

We design PI-type feedback controllers [20, 21] to allow for QoS control. These controllers tune the CPU frequency towards achieving the target QoS levels in response to phase variations within the applications as well as potential changes in the target QoS levels during runtime.

4.2 Fine-grained DVFS State Scheduling

Our earlier work [21] has shown that, in addition to controlling the average processor frequency to control QoS, fast frequency switching and temporal distribution of high frequency states within short intervals can improve thermal-efficiency. We illustrate this principle in Figure 6. Due to thermal time constants [10, 16], temperature responds gradually to changes in frequency/power levels. As shown in Figure 6b, this phenomenon implies that breaking the high frequency operation into short intervals and temporally distributing them over time can minimize the peak temperature while delivering the same average frequency. We implement this observation into our platform (MDP8974) as a kernel-level DVFS state scheduler unit with a *sysfs* interface to allow user programs to easily enable/disable or assign new target frequency levels. The target average frequency for the DVFS state scheduler is determined by the user-level feedback controller as explained in Section 4.1. An analytical proof of this technique for minimizing the peak temperature is available in our work [21]. Prior work has explored the benefits of quick DVFS in thermal simulation environments [10, 16] while we have shown its effectiveness on a real system for the first time. One critical limitation of temperature minimization via fast DVFS switching on a real hardware is the performance overhead associated with DVFS state transitions. This case is pointed out in Figure 7 which shows the measured (from the OS layer) performance overhead over different switching granularities. Performing DVFS faster than every 5ms can incur as much as 25% performance overhead.

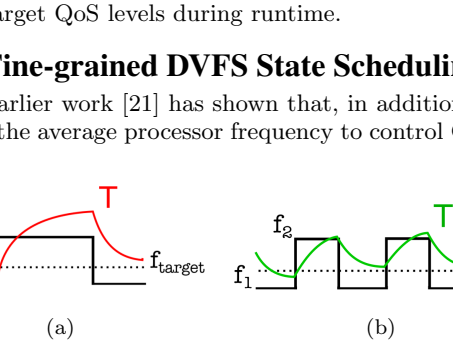


Figure 6: The key idea of the thermally-efficient DVFS state scheduler. By performing quick frequency switching and temporally distributing the states over time (right), the peak temperature can be reduced.

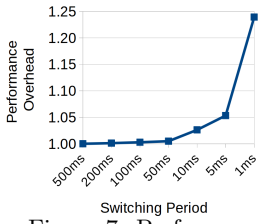


Figure 7: Performance overhead of DVFS.

4.3 Exploiting HW/SW level Heterogeneity for Thermal Efficiency on Heterogeneous CPUs

Our recent work [20] demonstrates that mobile applications are driven by a relatively few number of highly *QoS-critical* threads that determine the performance. We exploit this novel observation to perform thermally-efficient thread-to-core mappings on heterogeneous multicore mobile processors. Specifically, as opposed to treating all threads equally in scheduling decisions [24], we propose to restrict the high-performance cores (HW-level heterogeneity) only for accelerating the QoS-critical threads (SW-level heterogeneity) . We leverage the low-power cores in the system for executing the other *non-critical* application threads. Such criticality-driven task scheduling significantly reduces temperatures due to reduced load on power-hungry cores while still providing similar performance [20]. We detect such QoS-critical threads via a simple offline characterization process in which we measure the QoS improvement when an individual thread is moved from a low-performance core onto a high-performance core. Consider the case in Figure 8 [20]. The figure indicates, at two different frequencies, the QoS improvement when the threads of the *Rock Player* multimedia player application are incrementally moved to high-performance A15 CPU cluster. QoS sharply increases to the peak level after the 57th thread is executed on a high-

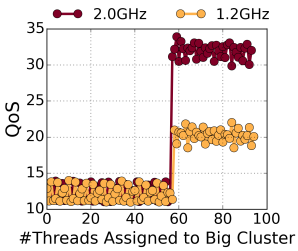
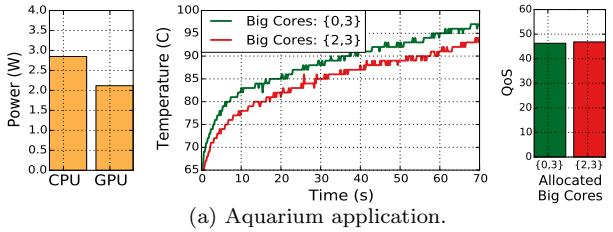
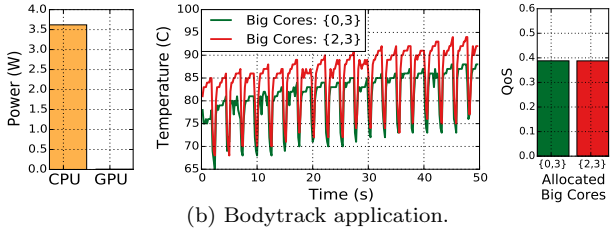


Figure 8: *Rock Player*'s QoS-critical thread [20].



(a) Aquarium application.



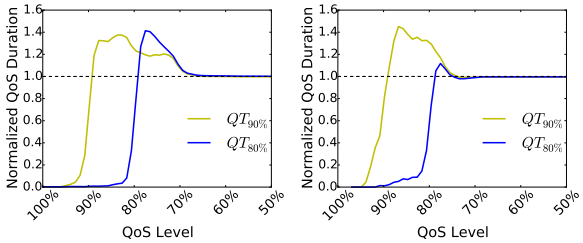
(b) Bodytrack application.

Figure 9: Power breakdown (left), temperature traces (middle) and QoS (right) for *Aquarium* and *bodytrack* under different big core assignments ({0,3} and {2,3}) [20].

performance core. This QoS-critical thread by itself dictates the overall performance for this application. The default HMP scheduler [3] for current heterogeneous mobile CPUs disregards such information and greedily assigns the threads with relatively high CPU usage to power-hungry cores for performance. This results in higher CPU usage (26% for *Rock Player*) on power-hungry cores without any significant improvements in QoS and leads to quick elevation of temperatures. We apply the same characterization to all applications to identify the critical threads and communicate this information to runtime thread mapper.

4.4 CPU-GPU Thermal Coupling Aware Selection of Thermally-Efficient Cores

Traditionally, thermal management policies have considered CPU in isolation from the other on-chip components. As mobile SoCs integrate high-power GPUs, however, thermal coupling between GPU and CPU cores have become an emerging phenomenon [19, 20]. In our work [20], we observe that, depending on the applications' graphics processing demand, the thermal efficiency of cores can alter significantly. Figure 9 [20] illustrates this observation through measurements on a Odroid-XU3 platform. We show the temperature profiles of two applications with distinct CPU and GPU usage when their two highest utilization threads are assigned to cores {0,3} and {2,3}. The allocation {0,3} achieves the lowest temperature for *bodytrack* while it leads to the most heating among all possible allocations when the GPU is highly utilized as in the *Aquarium* application. {2,3} results in the worst peak temperature for *bodytrack* while achieving lower temperature than {0,3} for *Aquarium*. The GPU causes quick heating when its highly utilized and nearby CPU cores (i.e., core 0) are used for execution. When lightly utilized, however, GPU can act as a head spreader for the CPU cores. We capture this interplay between the thermal efficiency of the CPU cores and GPU usage through an exhaustive offline characterization process using CPU and GPU microbenchmarks. Specifically, we rank the CPU cores according to their thermal efficiency under varying levels of GPU power dissipation. During runtime, we monitor application's GPU usage as a proxy for CPU-GPU thermal coupling and use the offline information to determine the most thermally-efficient CPU cores for executing the QoS critical threads.



(a) H264 (b) Bodytrack

Figure 10: Normalized duration of time spent above a QoS level by the proposed QoS tuning + DVFS state scheduler policy for different target QoS level. "$QT_{X\%}$" represents the proposed policy with X% QoS goal [21].

5. EVALUATION

In this section, we evaluate the improvements in sustained performance durations provided by our techniques for both homogeneous and heterogeneous mobile processors. Our runtime management technique for homoegeneous mobile CPUs integrates QoS tuning with DVFS state scheduling; while our policy for heterogeneous mobile CPUs integrates QoS tuning with CPU-GPU thermal coupling and thread criticality awareness. We evaluate our thermally-efficient QoS management approach under both processor and skin temperature constraints to demonstrate its benefits under multiple thermal constraints of modern smartphones as explained in Section 3. For experiments under CPU thermal constraints, we use Odroid-XU3 with a big.LITTLE CPU architecture and MDP8974 smartphone with a homogeneous CPU. For evaluation under skin temperature constraints, we use the MDP8974 smartphone.

Extended sustainability under CPU thermal constraints. Figure 10 [21] presents the improvements over the baseline with the proposed QoS tuning policy (named as QT) for homogeneous CPUs on a MDP8974 platform under CPU temperature constraints. The figure shows the duration of time spent above a QoS level (x-axis) using QT as normalized to the baseline (ondemand governor [1]). We experiment with two target QoS levels which are set to 90% and 80% of the maximum QoS for an application. While the baseline ondemand governor greedily converts the thermal into performance by selecting high DVFS states, our technique provides 'just enough' yet sustainable QoS by restricting the short term performance to target QoS levels and provides substantially longer execution time around the given QoS targets. For *bodytrack* and *h264* under 90% target QoS, QT provides 47% and 38% longer duration above this target QoS, respectively. Relatively less (11%) gain in sustainability is achieved for *bodytrack* under 80% QoS target. This is because the QoS degrades below 80% with the baseline policy only for a short duration.

Next, we demonstrate our achievements in sustainable QoS on a heterogeneous multi-core CPU available in Odroid-XU3 [4] platform. Our QoS management strategy (named as QScale) effectively minimizes the thermal density of the high-power A15 cores by restricting their use to only highly QoS-critical threads and selecting the most thermally-efficient cores for execution. Figure 11 [20] plots the sustained QoS durations for 3 policies (QScale versus two baselines) under different QoS targets. The *DVFS-only* policy represents the class of policies that rely only on DVFS for QoS control (i.e., without criticality and coupling awareness). Our policy

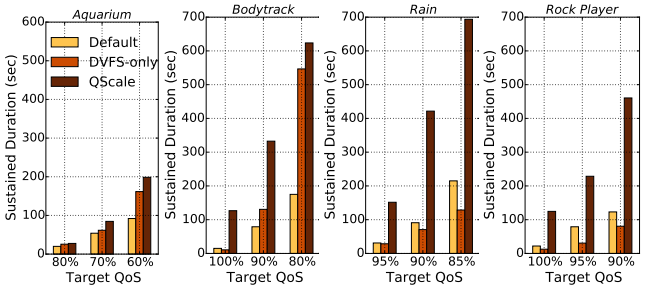


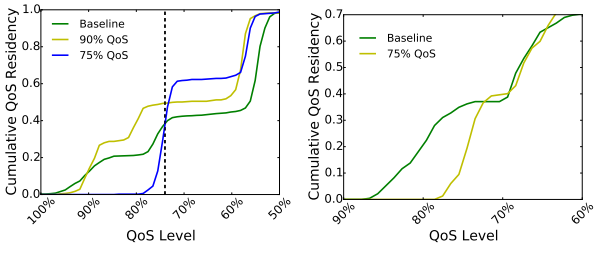
Figure 11: Sustained QoS durations with default management (*Android's Interactive Governor [1] + ARM HMP scheduler [3]*), *DVFS-only* and our QScale policy [20].

consistently provides the longest durations of sustained QoS for all four applications. Remarkably, while the two baseline policies are not able to provide the peak (100%) QoS for more than 10 seconds for the *bodytrack* (computer vision application) and *Rock Player* (video player) applications due to throttling, our QScale policy sustains the peak QoS for around 2 minutes by leveraging the criticality information.

Extended sustainability under skin level thermal constraints. We show the benefits of our thermally-efficient QoS tuning approach under skin temperature constraints by running two graphics applications, *Aquarium* and *Pearl Boy*, for 15 minutes on MDP8974 with a homogeneous CPU. Figure 12 [21] plots the cumulative QoS distribution. 100% QoS corresponds to 40 FPS for *Aquarium* and 60 FPS for *Pearl Boy*. Sustained QoS duration improves by 9% (from 36% to 40%) for *Pearl Boy* under 75% QoS limit. For *Aquarium*, our policy with 75% target QoS (30 FPS) improves the duration above this target by 55%, from 40% of the overall execution to 62%. The dashed 30 FPS line corresponds to bare minimum user tolerable QoS [25]. Thus, managing QoS temperature tradeoff while ensuring 'just enough' performance can provide the user with a 55% longer duration with an acceptable FPS level.

6. CONCLUSION

In this paper, we have shown that SoC and skin level thermal constraints can incur significant QoS degradations over extended durations as built-in power management policies greedily maximize QoS under temperature limits. We have discussed various practical runtime management solutions that can slowdown SoC and skin level heating while strictly adhering to minimum user requirements. We have shown effectiveness of our techniques on real-life devices in terms of mitigating thermal throttling and providing longer durations of sustainable performance.



(a) Aquarium (b) Pearl Boy

Figure 12: Cumulative QoS distribution for the two WebGL applications. Dashed line (left) represents 30 FPS limit [21].

7. REFERENCES

[1] Android cpu governors. http://goo.gl/Mlr9U1.

[2] Aquarium. http://webglsamples.org/aquarium/aquarium.html.

[3] Heterogeneous multi-processing for big.little. https://www.arm.com/files/pdf/Heterogeneous_Multi_Processing_Solution_of_Exynos_5_Octa_with_ARM_bigLITTLE_Technology.pdf.

[4] Odroid-XU3 Mobile Development Board. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127.

[5] Pearl Boy. https://www.chromeexperiments.com/experiment/pearl-boy.

[6] Rain. https://codepen.io/Sheepeuh/pen/cFazd.

[7] Revisiting SHIELD Tablet: Gaming Battery Life and Temperatures. [online] Available: http://www.anandtech.com/show/8329/revisiting-shield-tablet-gaming-ux-and-battery-life.

[8] When benchmarks aren't enough: Cpu performance in the nexus 5. [online] http://arstechnica.com/gadgets/2013/11/when-benchmarks-arent-enough-cpu-performance-in-the-nexus-5/.

[9] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, 2008.

[10] T. Chantem, X. S. Hu, and R. P. Dick. Online work maximization under a peak temperature constraint. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 105–110, 2009.

[11] B. Egilmez, G. Memik, S. Ogrenci-Memik, and O. Ergin. User-specific skin temperature-aware dvfs for smartphones. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1217–1220, 2015.

[12] M. Halpern, Y. Zhu, and V. Reddi. Mobile cpu's rise to power: Quantifying the impact of generational mobile cpu design trends on performance, energy, and user satisfaction. In *IEEE 22th International Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[13] J. L. Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[14] H. Hoffmann et al. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *International Conference on Autonomic Computing*, pages 79–88. ACM, 2010.

[15] H. Hoffmann et al. Dynamic knobs for responsive power-aware computing. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[16] M. A. Khan, C. Hankendi, A. K. Coskun, and M. C. Herbordt. Software optimization for performance, energy, and thermal distribution: Initial case studies. In *International Green Computing Conference and Workshops (IGCC)*, 2011.

[17] T. Lanier. ARM Cortex-A15 processor. http://www.arm.com/files/pdf/AT-Exploring_the_Design_of_the_Cortex-A15.pdf.

[18] R. Pozo and B. Miller. Scimark 2.0. http://math.nist.gov/scimark2/.

[19] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel. Improving mobile gaming performance through cooperative cpu-gpu thermal management. In *Proceedings of the 53rd Annual Design Automation Conference (DAC)*, 2016.

[20] O. Sahin and A. Coskun. QScale: Thermally-efficient QoS management on heterogeneous mobile platforms. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. ACM, 2016.

[21] O. Sahin, P. Varghese, and A. Coskun. Just enough is more: Achieving sustainable performance in mobile devices under thermal limitations. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. ACM, 2015.

[22] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras. Predictive dynamic thermal and power management for heterogeneous mobile platforms. In *DATE*, 2015.

[23] Q. Xie, M. J. Dousti, and M. Pedram. Therminator: a thermal simulator for smartphones producing accurate chip and skin temperature maps. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 117–122. IEEE, 2014.

[24] J. Yun, J. Park, and W. Baek. Hars: A heterogeneity-aware runtime system for self-adaptive multithreaded applications. In *DAC*, pages 107:1–107:6, 2015.

[25] Y. Zhu, M. Halpern, and V. J. Reddi. Event-based scheduling for energy-efficient QoS (eQoS) in mobile web applications. In *HPCA*, pages 137–149, 2015.