Just Enough is More: Achieving Sustainable Performance in Mobile Devices under Thermal Limitations

Onur Sahin, Paul Thomas Varghese, Ayse K. Coskun
Electrical and Computer Engineering Department, Boston University, Boston, MA, USA
{sahin, vpaul, acoskun}@bu.edu

***Abstract*—With the integration of high-performance multicore processors and multiple accelerators into modern mobile system-on-chips (SoCs), power densities have grown substantially. As a result, thermal management policies, which ensure operation at thermally safe conditions, became essential components of state-of-the-art mobile systems. Traditional thermal throttling approaches aim at maximum utilization of the available thermal headroom to minimize the performance loss and maximize user performance. This paper demonstrates that, in a mobile platform, such greedy techniques can lead to significant degradation in the quality-of-service (QoS) levels as the duration of device activity increases, leading to inconsistent user experience over time. We demonstrate that incorporating user/application QoS requirements into mobile power management to provide "just enough" performance (instead of always maximizing performance) allows for more efficient usage of the thermal headroom, which translates to substantially extended durations of sustainable performance. We propose a closed-loop QoS control policy, including an efficient dynamic voltage and frequency scaling (DVFS) state scheduling technique, to minimize the thermal impact for extending the *sustainability of desired QoS levels*. Experiments on a modern smartphone show that the proposed technique provides up to 74% longer sustainable performance while meeting the target QoS demands for a variety of real-life applications.**

I. Introduction

Mobile devices have been growing in popularity and have become essential parts of our daily lives. Growing user demand for faster processing and enhanced visual quality have been fulfilled by the increased computational capabilities of state-of-the-art mobile devices. Modern mobile system-on-chips (SoCs) have grown in complexity and processing power by integrating high-performance multicore processors and various accelerators (Graphics Processing Unit, Digital Signal Processor, etc.), resulting in excessive power densities. Higher power densities lead to elevated temperatures, which downgrade device reliability [2] and energy efficiency due to increased leakage power [17]. Incorporating multiple heat generating components such as battery, display and CPU into a small form-factor device with limited cooling makes maintaining safe chip temperatures even more challenging. Thus, modern mobile systems adopt CPU throttling techniques that adaptively reduce the operating frequency of the mobile processor to mitigate thermal emergencies. Thermal throttling, however, incurs performance drops and may impair user satisfaction in mobile systems where a satisfactory QoS level should be delivered to the user.

Existing power and thermal management techniques in mobile devices greedily exhaust the available thermal headroom to improve performance in case of increased computational demand. Although this approach works well for improving the QoS in relatively short applications, rapidly elevated temperatures can significantly increase the performance impact of throttling as the durations of the mobile device (phone, tablet, etc.) activity get longer. More aggressive thermal throttling induces larger performance degradations and leads to inconsistent performance levels during the application run. While current power management techniques in mobile devices favor short term performance, mobile system users also demand consistent performance for the applications that run for minutes or longer (i.e., consistent frame rate in gaming/video streaming [21] or webpage rendering time in browsing [30]). In fact, users have already reported significant performance drops in new generation high performance smarphones during extended durations of use [4][14]. Therefore, power management solutions in mobile devices need to address *performance sustainability*, as opposed to traditional short term performance oriented design, in face of the growing thermal limitations.

This paper addresses the problem of providing the user with sustainable performance levels over extended durations of mobile system use. We present real-life experiments to demonstrate the impact of thermal limitations on achieving performance sustainability using a modern smartphone platform. We observe that the available thermal headroom could be utilized more efficiently for longer durations by restricting the short term performance to a level that is "just enough" to meet the user's QoS demand. Based on this observation, we propose a novel QoS tuning technique that takes the target QoS goal as the primary performance constraint and attempts to sustain this target for the maximum duration by efficiently tuning the processor power level.

Our work makes the following specific contributions:

- We demonstrate the potential behind trading off short term performance in mobile devices for minimizing heat generation and achieving extended durations of sustainable performance.
- We propose a closed-loop QoS control policy integrated with an efficient DVFS state scheduler that achieves fine-grained CPU power control for meeting a wide range of possible QoS targets with minimal thermal impact.

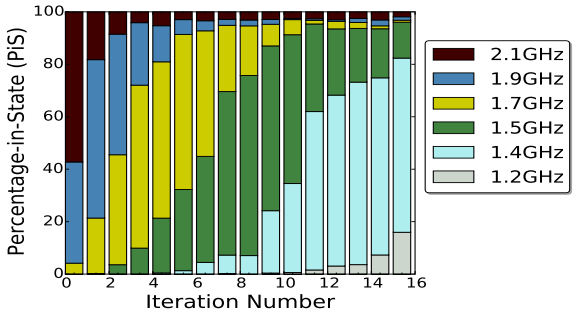


Fig. 1: Frequency residencies over time on a MDP8974 smartphone during continuous use. Performance impact of throttling increases over time as the CPU is forced to use lower frequencies to meet the thermal constraints.

- We evaluate our approach under both processor and skin temperature constraints using a variety of real-life applications with different QoS metrics. Our experiments on a modern smartphone show up to 74% longer durations of sustainable performance using the proposed technique.

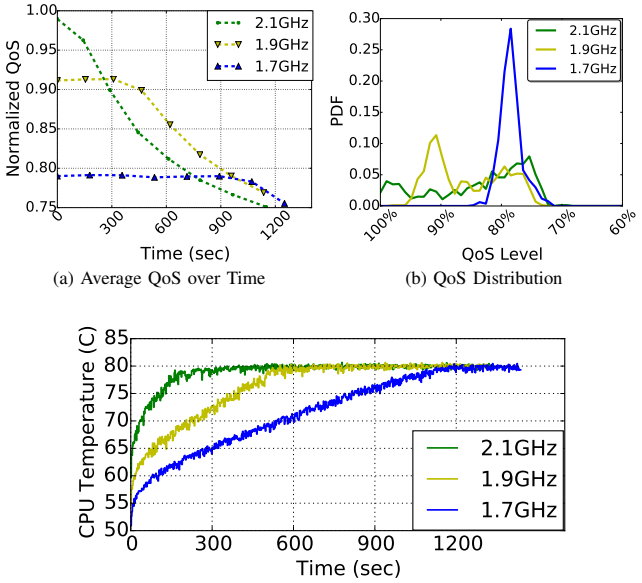
The rest of this paper starts by motivating the need for QoS-awareness in mobile thermal management to improve performance sustainability. Section III provides an overview and also the design details of our QoS tuning framework. Section IV presents the experimental methodology and integration on a real smartphone platform. Section V evaluates our technique. Section VI discusses the related work and Section VII concludes the paper.

II. Motivation

To exemplify the impact of extended application durations on sustained performance, Figure 1 shows the frequency residencies when the FFT application from Scimark Java benchmark suite [22] is repeatedly run on a Qualcomm Snapdragon MDP8974 smartphone [23]. The baseline CPU frequency scaling governor[1] scales the frequency to the highest level to boost performance and, initially (iterations 1-2), the application is able to operate below the thermal limit by throttling down to lower two frequencies only (1.9-1.7GHz). It should be noted that, in this example, frequencies lower than 2.1GHz level are enforced due to thermal throttling rather than by the power management scheme. In the later iterations, there is a clear shift towards utilizing lower frequencies due to more aggressive throttling applied by the baseline thermal management policy[1], which significantly reduces performance over time. For instance, in the last iteration, more than 80% of the running time is spent at 1.4GHz and 1.2GHz, while the application was well able to run without scaling down to those two frequencies initially. This example illustrates that *greedily utilizing the thermal headroom to boost short term performance can lead to significant performance loss over extended durations*.

We present another experimental scenario to point to the potential trade-off between the instant short term performance and sustainable performance. Figure 2 presents an experiment

[1]In the given experiment, default ondemand governor is used. The baseline thermal throttling policy is a PID controller with 80 °C thermal set-point that operate hierarchically with the ondemand governor.



(a) Average QoS over Time (b) QoS Distribution

(c) CPU Temperature Trace

Fig. 2: An illustration of the effect of trading off the short term performance on performance sustainability. The experiment corresponds to a repetitive run of bodytrack application [7] at 3 static frequency settings and QoS values are normalized to maximum QoS.

that corresponds to a repetitive run of the bodytrack application [7] at three different static frequency levels. Note that the system can still throttle the frequency below the assigned static level to avoid thermal violation. Figure 2a shows the average QoS for each iteration of the application over time. The maximum static frequency setting, 2.1GHz, gives the highest QoS initially. The QoS level, however, sharply reduces after the CPU reaches its thermal limits at around 220 seconds, as shown in Figure 2c. The QoS level continues to downgrade as a result of more aggressive thermal throttling and, at the end of the execution, QoS degrades to 25% of the initial maximum. The QoS drops below 90%[2] at around 300 seconds when using the aggressive 2.1GHz setting, while setting the frequency at 1.9GHz frequency allows to sustain the QoS level above 90% for 450 seconds. Figure 2b shows the QoS distribution for this experiment. The highest power setting results in wider distribution of the QoS while the 1.9GHz setting is able to rein the QoS distribution towards the 90% range (indicating longer duration of execution time spent around the 90% QoS). These results indicate that *lowering the short term performance requirements to "barely" meet the target QoS level, can enable longer sustainability of desired QoS goals*.

III. QoS Tuning Framework

In this section, we introduce our runtime framework and policies for efficiently tuning the QoS to "barely" match the target levels in the pursuit of achieving longer durations of sustainable performance. Figure 3 presents an overview of the framework that we have integrated into our platform. Our design comprises of three main components. The *closed-loop controller* takes the runtime tunable QoS level as a

[2]We choose 90% as an example acceptable QoS level to explain our motivation

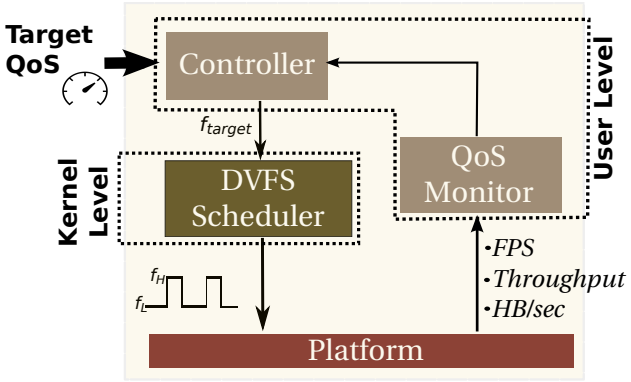


Fig. 3: Overview of the implementation framework. Frames per second (FPS), throughput and heartbeat per second (HB/sec) correspond to the QoS metrics for our applications that are listed in Section IV-B.

performance target and determines the operating frequency of the processor. The *DVFS scheduler unit* converts the continuous target frequency provided by the controller into a time-scheduled sequence of available discrete DVFS levels to efficiently match the continuous target frequency with minimal thermal impact. The *QoS monitoring unit* periodically monitors the frames-per-second (FPS), throughput, or heartbeats/second (HB/sec). We leave the discussion on platform integration of this framework to Section IV-C, and proceed with the implementation details of the controller and the DVFS scheduler in the following sections III-A and III-B, respectively.

A. Closed-loop QoS Controller

We design a feedback controller for dynamically adjusting the CPU speed to converge QoS towards desired levels. The controller tracks the progress of the application towards the target QoS by interacting with the *QoS Monitoring Unit* at every control interval. We use the following performance model, which represents the QoS level for the next time interval ($Q[k+1]$) as a fraction of the maximum achievable QoS (Q_{max}) at the highest frequency setting:

$$Q[k+1] = Q_{max}u[k] \tag{1}$$

$$e[k] = Q_{target} - Q[k] \tag{2}$$

The control signal $u[k]$ ranges between 0 to 1 and corresponds to the frequency scaling factor which determines the QoS level at time $k+1$. Since the goal of the controller is to minimize the difference between the target and current QoS levels, the error term $e[k]$ simply corresponds to this difference. The transfer function of the Equation 1 in the z-domain is given by:

$$F_1(z) = \frac{Q(z)}{U(z)} = \frac{Q_{max}}{z} \tag{3}$$

We find the transfer function $F_2(z)$ that defines the correspondence between the control signal and the error term by setting the following global transfer function of the closed-loop control system to $1/z$:

$$G(z) = \frac{F_1(z)F_2(z)}{1 + F_1(z)F_2(z)} \tag{4}$$

We obtain the discrete-time representation of the controller equation by substituting the $F_2(z)$ and taking the inverse z-transform as follows:

$$F_2(z) = \frac{U(z)}{E(z)} = \frac{z}{Q_{max}(z-1)} \tag{5}$$

$$u[k+1] = u[k] + e[k]/Q_{max} \tag{6}$$

We examine the stability and convergence of this control system by analyzing the global closed-loop transfer function $G(z)$ in z-domain. The closed-loop transfer function of $1/z$ has only one pole located at zero, which lies within the unit circle, ensuring the stabilization around the target QoS. Convergence to the target QoS level can be examined by evaluating the $G(z)$ at $z = 1$ and verifying a unit gain. Since $G(z) = 1/z$ evaluates to 1 at $z = 1$, the system has unit gain at the steady state and converges to the target QoS. The settling time of the controller is a function of the largest pole (a) of the closed-loop transfer function [12], approximated by $-4/log(a)$. Since the $G(z)$ has its single pole located at 0, the controller can converge instantly, limited by the controller invocation period in practice.

B. DVFS State Scheduler

The controller provides a continuous output signal while the CPU can only support discrete DVFS levels. We propose a DVFS state scheduler which divides the controller period into bins and switches between two neighbor frequency levels to produce an average frequency that matches the controller output. We also demonstrate the potential to minimize the thermal impact by making thermally-aware scheduling decisions to further extend the durations of sustainable performance.

Minimizing the Thermal Impact with Efficient DVFS Scheduling. We use the following discretized version of a lumped RC thermal model similar to prior research [25] to demonstrate the intuition behind our scheduling approach for minimizing the peak temperature of the CPU:

$$T[n+1] = T[n] + S(R_{th}P_k[n] - T[n])/R_{th}C_{th} \tag{7}$$

$$T[n+1] = c_1T[n] + c_2\hat{T}_k \tag{8}$$

where S is the sampling period, R_{th} and C_{th} are the thermal resistance and capacitance, $T[n]$ is the temperature at sampling interval n, $P_k[n]$ is the power level corresponding to DVFS state k, and $\hat{T}_k = R_{th}P_k[n]$ is the steady-state temperature for the power level P_k.

Consider the case where the scheduler estimates M bins to be scheduled with the higher frequency state in the following control interval, and those high frequency states are applied at distances of L. Next, we show that scheduler can reduce the peak temperature by increasing the distance L. Using Equation 8, we write the peak temperature at the end of the M^{th} high frequency state as follows:

$$T_p = c_1^{ML} T[0] + c_2 \sum_{i=0}^{M-1} c_1^{iL} \hat{T}_h + c_2 \sum_{i=0}^{(M-1)} \sum_{j=iL+1}^{(iL+L+1)} c_1^{j} \hat{T}_l \quad (9)$$

Since c_1 and c_2 are less than zero, when the distance between the high frequency states (L) is increased, the last term dominates and temperature approaches lower steady state temperature $\hat{T}_l$. Thus, distributing the high frequency states furthest from each other reduces the increase in temperature. Based on this intuitive observation, our scheduler implements maximum spatial distribution of the high frequency state bins within the control interval.

Impact of DVFS Granularity on Temperature. As a result of thermal time constants, temperature exhibits a "gradual" increase or decrease than a step thermal response. Thus, in addition to efficient DVFS scheduling, applying the DVFS state decisions faster, or increasing the number of switches within the interval, can also reduce maximum temperature due to the thermal buffer provided by the thermal time constants [9]. We exploit such benefits of the fast DVFS within the limitations of our experimental platform as pointed out in Section IV-C.

IV. Experimental Methodology

A. Target Platform

Our target experimental platform is a state-of-the-art Qualcomm Snapdragon MSM8974 smartphone [23] that hosts a Snapdragon 800 SoC (used in many modern smartphones, e.g., Nexus 5 and Galaxy S4). The Snadragon 800 SoC consists of a Quad Core Krait 400 CPU along with an Adreno 330 GPU, 2GB LPDDR3 RAM and is powered by a 1,600mAh Li-ion battery. The phone runs Android KitKat version 4.4.2 and Linux 3.4.0 kernel. The Krait 400 CPU supports 12 operating frequencies ranging from 300MHz to 2.1GHz. Temperature measurements can be done on a per-core basis via on-chip thermal sensors. Sensor readings for the CPU cores, battery and skin temperature are performed using the thermal virtual file system provided by the Linux kernel (i.e., /sys/class/thermal) with $\pm 1^{\circ}$C accuracy. We use the *logcat* system debugging tool available as part of the Android framework for monitoring the frames per second and use *perf_event* kernel API for accessing hardware performance counters. Our phone allows for measuring only the overall power consumption using the voltage and current sensors. For the CPU applications that do not require graphical interface, we turn-off the LCD display throughout the measurements. We leave the LCD display on for the GPU applications.

B. Application Set

Mobile systems run a broad range of applications and a single performance metric cannot gauge performance of all applications. Thus, we construct a benchmark set for our experiments by combining applications from various domains and evaluate them using different QoS metrics as summarized in Table I. The LU application, a common kernel in many image/video processing and mobile healthcare applications, is selected from Scimark 2.0 [22], which is a benchmark suite for testing Java based platforms. A video encoding (H.264) and an artificial intelligence application (Sjeng) are chosen from the SPEC CPU2006 [13]. We use two online graphics applications created with WebGL, Aquarium [1] and Pearl Boy [3]. The Aquarium shows an animation of fishes in a tank, while the Pearl Boy is an interactive application requires directing a boat in the sea. To ensure consistency between the runs, we automate the user interaction by applying the same sequence of *input swipe* commands for each experiment through a lightweight background shell program. We also use Heartbeats [15] instrumented version of the bodytrack computer vision application from the PARSEC suite [7]. Heartbeat framework allows to monitor application-specific QoS using a standardized interface and, for the bodytrack application, this framework emits a heartbeat whenever the processing of one scene is completed. Since the Heartbeats framework could be applied to a wider domain of applications for QoS monitoring and tuning purposes, we find value in demonstrating the applicability of our techniques on a representative Heartbeat-instrumented application.

Application	Category	QoS Metric
Sjeng	Artificial Intelligence	Throughput
H.264	Media Processing	Throughput
LU	Math	Throughput
Pearl Boy	Graphics/WebGL	Frames per second
Aquarium	Graphics/WebGL	Frames per second
Bodytrack	Computer Vision	Heartbeats/sec

TABLE I: Summary of applications and respective QoS evaluation metrics.

C. Baselines and Implementation Strategy

In this section, we provide the implementation details of our framework and summarize the baseline policies that we have used in our platform for comparisons against our policy.

Baseline Policies. The default CPU frequency scaling policy in our phone (and in most state-of-the-art Android devices) is the *ondemand governor* [20], which adjusts the CPU frequency based on the CPU load. Thus, we use the ondemand governor as our baseline power management scheme in our experiments.

Thermal throttling policies operate hierarchically with the CPU frequency governors and assign maximum frequency limits for ensuring operation below a thermal set-point. The CPU governors cannot use the frequencies that are above the assigned limit. Since the control-theoretic thermal management solutions are among the most commonly used techniques for maintaining the maximum temperature at a given threshold, we use a DVFS-based PID controller as the baseline CPU throttling mechanism. Modern smartphones also incorporate skin temperature management policies to keep the outer device temperature within the human comfort levels. Thus, performance degradations can occur due to increased skin temperatures as well. Since our MSM8974 device does not provide a skin temperature management policy by default, we implement the skin thermal management scheme available in the Nexus 5 smartphones. This policy assigns a maximum

		H264			Bodytrack			Sjeng			LU			
		ondemand & DTM	QT90	QT80	ondemand & DTM	QT90	QT80	ondemand & DTM	QT90	QT80	ondemand & DTM	QT90	QT80	QT70
Before Throttling	Average QoS	0.99	0.89	0.80	0.96	0.902	0.804	1.01	0.896	0.805	0.995	0.903	0.806	0.707
	Standard Deviation of QoS	0.035	0.042	0.037	0.02	0.028	0.043	0.086	0.059	0.065	0.107	0.109	0.075	0.074
Overall Execution	Average QoS	0.85	0.85	0.79	0.871	0.872	0.803	0.85	0.84	0.79	0.723	0.756	0.732	0.695
	QoS Degradation	27.2%	16.6%	**0.3%**	18.3%	11.3%	**0.1%**	28%	18%	4%	35%	30%	22%	7.4%
	Time Spent in Throttling	85.4%	55.5%	**6.8%**	59.4%	48.3%	**0%**	86.9%	61.7%	26.1%	87.7%	86.6%	73.9%	**45.4%**
	Average Power	0.99	0.97	0.88	0.92	0.97	0.86	0.99	0.96	0.89	0.97	1.02	0.97	0.91
	Energy Consumption	1.01	0.98	0.94	0.91	0.97	0.93	1.01	0.98	0.96	1.02	1.01	0.99	0.97
	QoS/Watt	0.99	1.02	1.05	1.09	1.04	1.08	0.99	1.013	1.03	0.997	0.994	1.006	1.03

TABLE II: A summary of results for the CPU applications. QT(X) represents proposed QoS tuning policy with X% target QoS. Average QoS, power, energy and QoS/Watt values are normalized to the highest static frequency setting (2.1GHz). QoS degradation corresponds to the percentage of QoS loss from the first to last iteration of the application run. Time spent in throttling results corresponds to the percentage of execution time thermal throttling is incurred.

Trip Point	Frequency Limit
40°C	1.9GHz
42°C	1.5GHz
44°C	1.2GHz

TABLE III: Temperature thresholds and target frequency limits of the baseline skin temperature controller.

CPU frequency limit whenever a skin temperature trip point is reached, as described in Table III. Both thermal throttling mechanisms poll the thermal sensors and assign frequency limits every 100ms to enable non-intrusive (less than %1 execution overhead) thermal management while maintaining sufficient time granularity to avoid thermal emergencies.

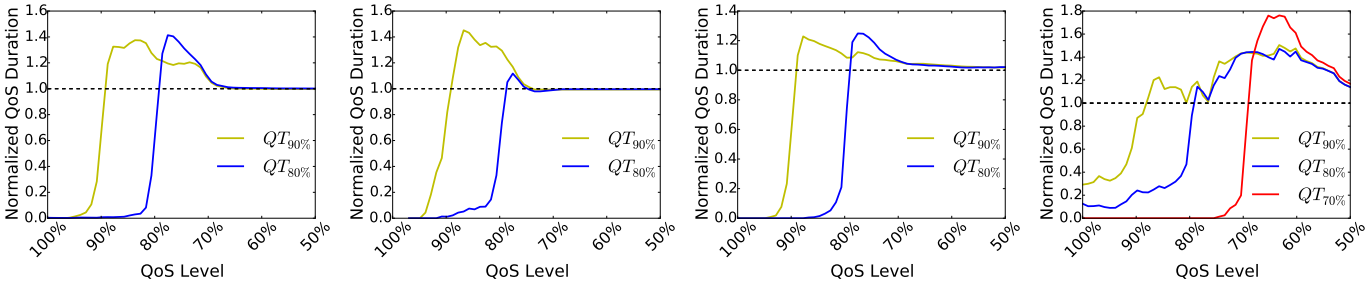
Implementation of the QoS Tuning Framework. We implement the closed-loop controller as a user-level program that regularly monitors the QoS level and passes the target frequency to the kernel-level DVFS scheduler. The controller is invoked every 200ms for the CPU applications and every 1 second for the GPU applications. We have observed noise in the FPS values when sampling at finer granularity. We implement our DVFS scheduler in the kernel-level as a new CPU governor with a *sysfs* interface to allow for assigning target frequency levels from the user space. The governor based implementation allows users to easily enable/disable our QoS tuning policy. The DVFS scheduler applies the frequency decisions at the granularity of 20 miliseconds via the *cpufreq* [8] interface. We have measured the frequency transition latency in our system to be 186.4 microseconds by wrapping the *cpufreq_driver_target* call in our kernel module with timing utilities. We have found 20 miliseconds to be the finest DVFS granularity that could be applied without introducing noticeable overhead ($<$1%) in our system. The maximum performance overhead of our framework is less than 1.3% across all the applications in our benchmark set.

V. Results & Evaluation

In this section, we present a thorough evaluation of the QoS tuning policy that we have proposed in this paper and demonstrate the benefits of our approach for achieving longer durations of sustained performance. We evaluate the CPU applications with the CPU temperature triggered dynamic thermal management policy (DTM_{cpu}), which is the PID controller based throttling scheme described in Section IV-C. For the graphics applications, we have observed that CPU temperatures did not reach to critical limits while the skin temperatures kept increasing over time. Thus, we also provide an evaluation of our QoS tuning policy with the skin temperature controller (DTM_{skin}) running as the throttling mechanism on our platform. We aim to show the benefits of our QoS tuning approach from the performance sustainability perspective under both processor and device-level skin temperature constraints.

Extended Sustainability with QoS Tuning on CPU applications. For the CPU applications, we explore three target QoS levels which are set to 90%, 80% and 70% of the average QoS achieved when the application is run at the highest static frequency setting on an initially cold system. For clarity, we do not present results for the 70% cases if the application QoS does not degrade to this level using the highest static frequency setting. We emulate extended application durations by repetitively running the applications for a fixed number of iterations. We determine the number of iterations based on a maximum battery temperature limit of $50\,^{\circ}\mathrm{C}$.

Table II gives a detailed overview of our experimental results where QT(X) corresponds to the proposed QoS tuning technique at the target QoS level of X%. All the values except for the QoS degradation and the time spent in throttling are normalized to the highest static frequency setting. We evaluate the phases of the execution without throttling (indicated by "before throttling" in Table II) separately to examine the controller's ability to meet QoS goals without the interference of the throttling policy. Overall, our controller is able to effectively meet the given QoS targets with less than 0.06 average deviation. In most cases, average QoS of the overall execution is lower than the "before throttling" phase due to the performance impact of the throttling. However, bodytrack and h264 applications are able to sustain the performance close to 80% QoS level throughout the whole execution as little or no thermal throttling is incurred at that level for these two applications. The highest QoS degradation is observed for the LU application, which spends 45% of time in throttling even in the lowest QoS target of 70%. This degradation is due to the power hungry nature of this CPU intensive computing kernel that quickly reaches the CPU thermal limits. For all benchmarks, the baseline *ondemand* policy continuously seeks



(a) H264 (b) Bodytrack (c) Sjeng (d) LU

Fig. 4: Normalized duration of time spent above a QoS level by the proposed QoS tuning policy for different target QoS level. "$QT_{X\%}$" represents the proposed QoS tuning policy with X% QoS goal. A data point in the figure corresponds to (Time spent above a QoS with QT)/(Time spent above a QoS with (DTM_{cpu}+ondemand).

to convert the thermal headroom into performance by scaling the frequency to high levels and incurs the highest QoS degradation due to the increased percentage of time spent in throttling. The QoS tuning policy with 90% target level achieves 38.6% reduction in throttling duration on average and consistently provides lower QoS degradation for all benchmarks. The QoS tuning also provides up to 14% and 7% reductions in power and energy, respectively.

Figure 4 presents the improvements in performance sustainability for our CPU applications. The figure shows the duration of time spent above a QoS level with the proposed QoS tuning (QT) policy as normalized to the baseline. The proposed technique provides substantially longer execution time around the given QoS target. This could be observed in Figure 4a,4b and 4c where the the curves start to rise significantly above the dashed line (normalized baseline) when approaching the the given QoS goals. For the h264, bodytrack and sjeng applcations, an average of 37% and 26.7% longer sustainability is achieved for the 90% and 80% QoS levels, respectively. Improvement by the QoS tuning on the bodytrack application for the 80% QoS level is lower (11%) as the QoS drops to 80% range for only a short duration of time with the baseline policy. The LU application, as shown in Figure 4d, provides the peak improvements in sustainability with 74% longer duration the 70% QoS target is sustained. This application has the highest power consumption among our applications and quickly reaches to thermal limits with the baseline setting. Therefore, using higher frequency settings results in higher QoS degradation for this application. In fact, the proposed policy is also unable to sustain the QoS around the target range for the higher 90% and 80% target levels and QoS distribution shifts towards a lower range.

Fine-grained QoS Control with DVFS scheduler. The DVFS state scheduler converts the continuous frequency targets into a time-scheduled distribution of two discrete frequency states that are neighbours of the continuous target frequency. This scheme enables fine-grained tuning of the QoS for accurately matching to the target levels. Figure 5 compares the proposed combined QoS controller and DVFS scheduler technique against two performance aware static frequency settings, 1.9GHz and 1.5GHz. We refer to such static frequency selection as performance aware since those two frequencies provide the nearest average QoS to 90% and 80% levels, respectively. Using the static frequency settings fails to precisely meet the target QoS goals. For instance, in Figure 5, 1.9GHz setting provides higher QoS than 90% goal while 1.5GHz setting provides 5% lower QoS than the target 80% level. Besides, since the static 1.9GHz setting provides higher QoS than the target level and consumes extra power for this offset, the QoS drops below the 90% level prior to proposed fine-grained QoS tuning technique due to earlier invocation of thermal throttling.

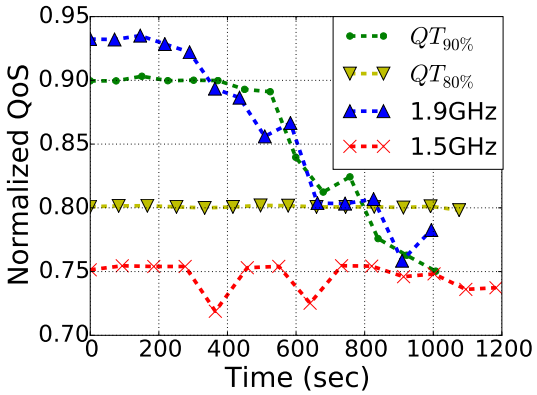


Fig. 5: Average QoS per iteration over time for the H264 application with the proposed QoS tuning and performance aware static frequency selection policy.

The DVFS state scheduler also aims to achieve the maximum spatial distribution of the higher frequency states to minimize thermal impact. Figure 6 shows the effect of such distributed scheme on the temperature trace of the h264 application. Undistributed scheme simply switches from low frequency state to high frequency state only once during the control period while the distributed policy applies finer granularity switching with maximum possible low frequency periods between the high frequency states, both providing the same average frequency. As annotated by the two arrows in Figure 6, the distributed policy allows for longer execution without reaching to the thermal limit.

Dynamic Adaptation to Changes in QoS Requirements. The target QoS requirements for an application could change during runtime for various reasons (upon user request, remaining battery level etc). Therefore, a good control policy should

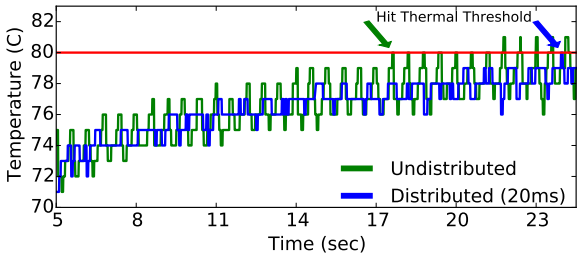


Fig. 6: Temperature traces for two DVFS scheduling schemes. Undistributed scheme simply switches from lower to higher frequency only once during the control interval (1 sec). Distributed scheme gains more thermal headroom by applying fine grain DVFS and scheduling high states farthest possible from each other. The duty-cycle is 33% high.

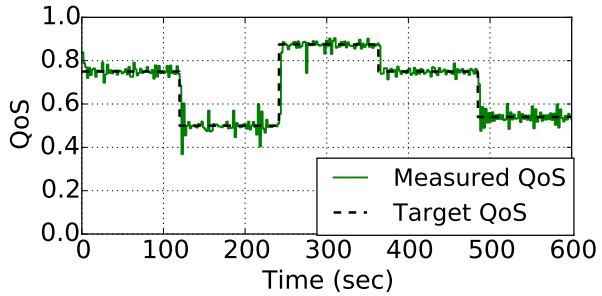
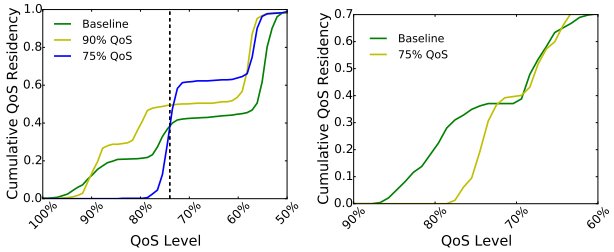


Fig. 7: QoS trace for the Aquarium WebGL application as the policy adapts to QoS requirements during runtime. The values are normalized to the maximum achievable FPS and QoS targets are arbitrarily modulated every 2 minutes.

respond to such changes in the performance requirements. In order to exercise our proposed framework with dynamically changing QoS goals, we design an experiment where the target QoS is arbitrarily modulated every two minutes during a 10 minute run of the aquarium graphics application. Figure 7 presents the QoS trace as the policy adapts to the changing target QoS levels. The closed-loop controller with DVFS scheduler enables fine-grained QoS tuning and is able to precisely meet the arbitrary QoS goals. We have measured the root mean square tracking error of our policy to be 0.088 in this experiment.

Performance Sustainability Under Skin Temperature Constraints. We further investigate the applicability of our motivation and QoS tuning technique under skin temperature constraints for extending the durations of target FPS levels using two graphics applications. Figure 8 shows the cumulative distribution of the QoS for both applications during a 15 minutes of continuous execution. 100% QoS corresponds to 40 FPS for the Aquarium application and 60 FPS for the Pearl Boy application. A data point corresponds to the fraction of the overall execution time spent above the corresponding QoS level. For the Pearl Boy application, QoS tuning with 75% target level improves the sustainability by 9%, from 36% to 40% of the execution time spent above the target. We do not show the 90% case as the baseline policy provides a QoS range below 88%. Figure 8a shows the cumulative QoS distribution for the Aquarium application and the dashed line corresponds to the 30 FPS limit which is pointed by prior research to be the lowest frame rate in the user tolerable range [21][29]. The QoS tuning policy with 75% target (30 FPS) increases the



(a) Aquarium (b) Pearl Boy

Fig. 8: Cumulative QoS distribution for the two WebGL graphics applications. Dashed line in the left figure shows the 30FPS limit. The baseline policy corresponds to ondemand+DTM_{skin}.

sustainability of this QoS level from 40% of the execution time to 62%, *providing 55% longer duration that the user can be provided with an acceptable FPS level.*

VI. Related Work

Thermal management of multicore CPUs and MPSoCs is a well studied subject for conventional computer systems. Control theoretic DVFS techniques provide effective temperature control while maximizing performance [6][25] and predictive techniques (e.g., [28]) have been applied to project thermal emergencies for minimizing temperature violations. There has been a recently growing effort towards thermal modeling and analysis of the mobile devices in particular. Xie et al. propose a resistance network based thermal simulation framework for obtaining component level steady-state temperatures [26] and derive an RC model of the thermal coupling between the battery and the application processor [27]. Singla et al. [24] present a temperature prediction and power budgeting methodology for heterogeneous mobile SoCs and show power savings compared to fan-based and reactive policies. ARM's new *Intelligent Power Allocation* [19] scheme aims to maximize performance under thermally limited scenarios by shifting the power between the heterogeneous CPU cores and GPU based on the expected performance return. Unlike the previous work, we do not attempt to improve performance under temperature constraints. Instead, we consider the target QoS levels as performance-wise sufficient and aim to sustain that QoS level for maximum duration.

Power management for meeting performance goals have been studied for various computing platforms. Ayoub et al. [16] propose a DVFS management technique for meeting throughput requirements in a server system. Lo et al. [18] present PEGASUS, which utilizes the Intel's *Running Average Power Limiter* for enabling fine-grained CPU power tuning in web clusters to match query latency requirements. Kadjo et al. [5] reduce the QoS requirements in memory bound applications and achieve platform level power savings in a mobile system. Pathania et al. [21] propose a CPU-GPU power budgeting algorithm to meet a frames-per-second constraint in mobile games. While the above techniques do not consider the sustainability of performance targets and thermal impacts, a recent study [11] points to the performance measurement flaws that occur due power level differences in the boosting

mode and the throttling mode in Intel's TurboBoost enabled processors. Other recent work [29] proposes to trade-off QoS within the user tolerable range for energy minimization in event-based mobile web applications while we present the case for trading off the QoS for thermal headroom to achieve longer sustainable performance.

Several studies addressed the scheduling of discrete DVFS states with thermal considerations in real-time systems domain. Applying faster switching between the discrete DVFS levels have been formally shown to maximize the workload under a thermal threshold [9] and minimize the peak temperature [10] in hard real-time systems. Inspired by those techniques in real-time systems domain, we utilize DVFS scheduling to enable fine-grained CPU power tuning and meet the target QoS constraints with minimal use of the thermal headroom for improved performance sustainability.

Overall, our work differs from the prior research by the following aspects: (1) we address performance (un)sustainability in mobile devices that arises due to thermal limitations; (2) we show the performance drawbacks of existing thermal management approach due to the pursuit of favoring short term performance; (3) we propose tuning the power management policies in mobile systems to match the target QoS demands for creating efficient usage of thermal headroom, enabling extended durations of sustainable performance; (4) we run all experiments on real-life systems.

VII. Conclusion

In this paper, we addressed the drawbacks of greedily exhausting the thermal headroom to boost short term performance in mobile devices that actively grow in power densities under limited cooling capabilities. Through experiments on a real-life platform, we demonstrated that existing thermal management approach leads to considerably lower performance as the duration of device activity increases, as a consequence of the increasing performance impact of thermal throttling. Diminishing performance imperils the user-experience in thermally limited modern mobile devices that must guarantee satisfactory quality-of-service (QoS) levels. In order to meet the user demand in the face of thermal limitations, we propose to trade-off performance for thermal headroom while meeting target QoS goals in a mobile system. We present a runtime framework with a closed-loop QoS controller and a DVFS state scheduler that enables fine-grained power tuning. Our results show that, utilizing the proposed framework for tuning the user performance to a "just enough" level to meet the minimum QoS requirement, allows for more efficient usage of the thermal headroom and extends the durations of sustained QoS levels by up to 74%.

References

[1] Aquarium. https://www.chromeexperiments.com/experiment/webgl-aquarium.

[2] Failure mechanisms and models for semiconductor devices, JEDEC publication JEP122C. http://www.jedec.org.

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

[4] 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.

[5] R. Z. Ayoub et al. Os-level power minimization under tight performance constraints in general purpose systems. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design (ISLPED)*, 2011.

[6] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini. A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011.

[7] 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.

[8] D. Brodowski. Linux kernel cpufreq subsystem. *URL http://www. kernel. org/pub/linux/utils/kernel/cpufreq/cpufreq. html*.

[9] T. Chantem, X. S. Hu, and R. P. Dick. Online work maximization under a peak temperature constraint. In *ISLPED*. ACM, 2009.

[10] V. Chaturvedi, H. Huang, and G. Quan. Leakage aware scheduling on maximum temperature minimization for periodic hard real-time systems. In *IEEE 10th International Conference on Computer and Information Technology (CIT)*, June 2010.

[11] L. Emurian et al. Pitfalls of accurately benchmarking thermally adaptive chips. *Power (W)*, 5:10, 2014.

[12] J. L. Hellerstein et al. *Feedback control of computing systems*. John Wiley & Sons, 2004.

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

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

[15] H. Hoffmann et al. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *Proceedings of the 7th international conference on Autonomic computing*, pages 79–88. ACM, 2010.

[16] D. Kadjo et al. Towards platform level power management in mobile systems. In *International System-on-Chip Conference (SOCC)*, 2014.

[17] N. S. Kim et al. Leakage current: Moore's law meets static power. *Computer*, 36(12), 2003.

[18] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*, 2014.

[19] M. Muller. Power constraints: From sensors to servers. [online] http://www.hotchips.org/wp-content/uploads/hc_archives/hc26/HC26-11-day1-epub/HC26.11-1-Z-Keynote1-Power-epub/HC26.11.190-Keynote1-Power-Muller-ARM-MM-v8.pdf.

[20] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, volume 2, pages 215–230, 2006.

[21] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated cpu-gpu power management for 3d mobile games. In *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2014.

[22] R. Pozo and B. Miller. Scimark 2.0. *URL: http://math. nist. gov/scimark2*, 2000.

[23] Qualcomm Developer Network. [online] Snapdragon MSM8974 MDP. https://developer.qualcomm.com/snapdragon-mobile-development-platform-mdp, 2014.

[24] G. Singla et al. Predictive dynamic thermal and power management for heterogeneous mobile platforms. In *DATE*. IEEE, 2015.

[25] K. Skadron et al. Control-theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture (HPCA)*, Washington, DC, USA, 2002.

[26] Q. Xie, M. J. Dousti, and M. Pedram. Therminator: A thermal simulator for smartphones producing accurate chip and skin temperature maps. In *ISLPED*, 2014.

[27] Q. Xie et al. Dynamic thermal management in mobile devices considering the thermal coupling between battery and application processor. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013.

[28] I. Yeo, C. C. Liu, and E. J. Kim. Predictive dynamic thermal management for multicore systems. In *DAC*, 2008.

[29] Y. Zhu, M. Halpern, and V. J. Reddi. Event-based scheduling for energy-efficient qos (eqos) in mobile web applications. In *HPCA*, 2015.

[30] Y. Zhu and V. J. Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *HPCA*, 2013.