Power and Thermal Analysis of Commercial Mobile Platforms: Experiments and Case Studies

Ganapati Bhat[1], Suat Gumussoy[2], and Umit Y. Ogras[1]

[1]School of Electrical Computer and Energy Engineering, Arizona State University, Tempe, AZ

[2]IEEE Member

Email: gmbhat@asu.edu, suat@gumussoy.net, umit@asu.edu

Abstract—**State-of-the-art mobile processors can deliver fast response time and high throughput to maximize the user experience. However, high performance comes at the expense of larger power density, which leads to higher skin temperatures. Since this can degrade the user experience, there is a strong need for power consumption and thermal analysis in mobile processors. In this paper, we first perform experiments on the Nexus 6P phone to study the power, performance and thermal behavior of modern smartphones. Using the insight from these experiments, we propose a control algorithm that throttles select applications without affecting other apps. We demonstrate our governor on the Exynos 5422 processor employed in the Odroid-XU3 board.**

I. INTRODUCTION

Commercial mobile platforms need to support multiple application categories, such as games, navigation, social media apps, and video players. In order to deliver competitive performance while running these apps, state-of-the-art mobile processors integrate multiple powerful general-purpose cores, graphics processing cores (GPU), and accelerators [7, 23]. At the same time, mobile platforms are severely limited by thermal design power (TDP) due to their small size and lack of active cooling [6, 19, 24, 28]. Power dissipation increases not only the junction temperature on the chip but also the skin temperature of the platforms, which directly impacts the user satisfaction [5, 20, 21]. Therefore, mobile platforms must deliver the highest performance without violating thermal constraints to maximize user satisfaction [17, 22, 27].

Competitive performance and fast response times are enabled by using multiple resources and operating them at high frequencies. For instance, the interactive governor on Android devices sets the frequency to the highest value whenever it detects user interactions. While this choice maximizes the response time and frame rate, it can also lead to thermal violations [3, 13, 27]. Higher chip temperature has a cascading effect of increasing the skin temperature, thus degrading the user experience [5, 20, 21]. Therefore, mobile platforms manage the temperature using thermal governors to mitigate this problem [9, 11, 22, 27].

Thermal governors monitor the temperature at the critical hotspots on the chip. If the temperature of any hotspot exceeds the thermal limit, the governors react by decreasing the frequency of the processing elements. In extreme cases, the governors resort to powering the cores off to reduce the temperature of the device [1, 19]. However, these actions can lead to significant performance degradation [16, 25, 26, 30].

In this paper, we first present experimental case studies with Nexus 6P smartphone and most popular Android apps. This experimental study allows us to study the operation of the default thermal governors shipped with commercial phones. Our results show that thermal throttling degrades the performance by as much as 34% while running popular Android applications. Our empirical results also demonstrate the need for integrated thermal-frequency governors. Thermal governors throttle the whole system instead of selectively throttling the resources that increase the temperature. For example, if a background application increases the temperature, the governors decrease the frequency of all processors in the system. This choice degrades the performance of all foreground applications. Finally, the outputs of the thermal and frequency governors may contradict each other [27]. More specifically, frequency governors increase the operating frequency to ensure responsiveness, while thermal governors throttle frequencies to maintain the temperature within safe limits. Hence, we also propose an approach that can throttle select applications without affecting the performance of other apps in the system. We demonstrate our approach on the Odroid-XU3 [10] board that employs a big.LITTLE heterogeneous system-on-a-chip, since it provides more flexibility to modify the default governors.

This paper is a part of the DATE 2019 Special Session on "Smart Resource Management and Design Space Exploration for Heterogeneous Processors". The other two papers of this special session are: "Smart Thermal Management for Heterogeneous Multicores" [11] and "Design and Optimization of Heterogeneous Manycore Systems enabled by Emerging Interconnect Technologies: Promises and Challenges" [14]. The rest of the paper is organized as follows. Section II overviews the related work. In Section III, we perform a case study with the Nexus 6P phone to illustrate the behavior of default thermal governors. Section IV presents a control algorithm that can selectively throttle applications and demonstrates it on the Odroid-XU3 board. Finally, Section V concludes the paper with some future directions.

II. RELATED WORK

A significant amount of recent research effort has focused on developing algorithms for dynamic frequency management

144

under thermal constraints [1, 5, 22, 26, 27]. For instance, the work in [26, 27] proposes a closed-loop controller to maximize the quality of service (QoS) provided by apps under thermal constraints. The controller ensures that apps can sustain their target QoS for a longer duration when compared to greedy approaches. Similarly, the work in [22] considers a cooperative CPU-GPU governor for improving the performance of gaming apps. Authors in [5] first develop a skin temperature predictor for smartphones and then use it for user-specific skin temperature aware frequency and voltage selection. The algorithm proposed in [1] uses a temperature prediction to determine a power budget that does not violate the thermal constraint. Then, it uses a gradient search algorithm to reduce the frequency of resources that result in the minimum performance loss. While these algorithms consider dynamic frequency management under thermal constraints, they do not consider the problem of selectively throttling background apps without affecting the foreground apps.

III. EMPIRICAL STUDY OF THERMAL THROTTLING ON COMMERCIAL SMARTPHONE

A. Experimental Setup

We perform our empirical study on a Nexus 6P [12] smartphone that runs Android 7 OS. A commercial smartphone with actual form-factor is preferred over commonly used experimental boards since the form-factor and packaging affect the thermal behavior. Furthermore, it enables us to evaluate the temperature and power management governors shipped to the end users. The Nexus 6P smartphone houses a Qualcomm Snapdragon 810 system-on-a-chip (SoC) [23], which integrates four low-power Cortex-A53 cores, four high-performance Cortex-A57, and an Adreno 430 GPU. The phone also has sensors to measure the temperature at various points on the SoC, such as the chip package, memory, and flash memory. Among these, we measure the temperature at the chip package, since it is used by the default governors to make thermal management decisions. In contrast to evaluation boards that include power measurement sensors, the Nexus 6P phone does not include power measurement sensors. Therefore, we use a National Instruments data acquisition system [18] to measure the power consumption of the phone at a sampling rate of 1 KHz.

Applications under study: Using our setup, we evaluate the power, performance, and thermal behavior of the most popular apps on the Google play store. We present the results for five representative apps from the top 30 apps on the Google play store. More specifically, we choose two games, one shopping app, one video conferencing app and one social media app, since they are the most popular app categories [29].

We run each of these apps under two scenarios. First, we disable the default temperature governor to measure the baseline performance of each app. Then, we repeat the experiments by enabling the default temperature governor shipped with the phone. This controlled experiment allows us to measure the impact on performance due to thermal throttling. In the fol-

lowing section, we analyze the temperature and performance of each application.

B. Performance and temperature analysis of popular apps

Paper.io game: Paper.io is one of the top five games available in the Google play store. Figure 1 shows the behavior of the temperature of the phone when running the Paper.io game. We observe that there is a significant increase in the temperature when we disable thermal throttling. The package temperature reaches 50°C at the end of the run. Then, we enable the default thermal governor and re-run the game. The dashed red line in Figure 1 shows that the thermal governor can successfully control the temperature. However, it comes with a significant performance penalty of about 12 frames per second (*FPS*). To understand the behavior of the phone in more detail, we analyze the usage of different GPU frequencies in Figure 2. When there is no throttling, the GPU operates 32% of the time at 510 MHz and 15% of the time at 600 MHz, which are the two highest frequencies. Figure 2 clearly shows that throttling decreases the use of these frequencies significantly. More specifically, the use of 510 MHz and 600 MHz drop to zero, while the percentage of operation at 390 MHz increases sharply from 15% to 67%. Since the GPU spends noticeably more time at lower frequencies, the game suffers a drop in performance. More specifically, the frame rate drops from 35 *FPS* to 23 *FPS*, which corresponds to 34% degradation in performance, as summarized in Table I.

Stickman Hook game: Stickman Hook is another game that is in the top five apps available on the Google play store. Figure 4 shows the temperature profile of the phone while playing Stickman Hook with the default thermal governor and by disabling it. We observe that the temperature reaches significantly higher values, especially after running the application for 50 seconds, when thermal throttling is disabled.
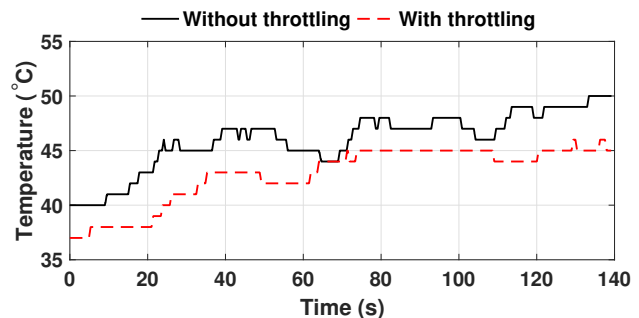

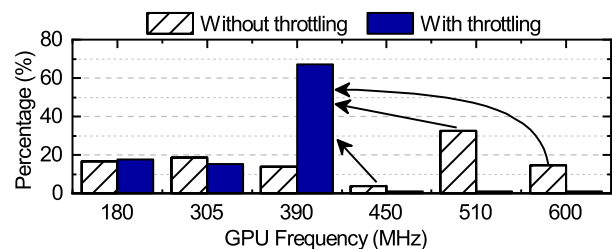Fig. 1: Temperature profile for Paper.io game.

Fig. 2: Usage of GPU frequencies in the Paper.io game

TABLE I: Median frame rate achieved while running popular Android apps with and without throttling, respectively.

| App | Frame Rate | | Percentage Reduction |
	Without Throttling	With Throttling	
Paper.io	35 *FPS*	23 *FPS*	34%
Stickman Hook	59 *FPS*	40 *FPS*	32%
Amazon	35 *FPS*	28 *FPS*	20%
Google Hangouts	42 *FPS*	38 *FPS*	10%
Facebook	35 *FPS*	24 *FPS*	31%

As expected, thermal throttling enables lower temperatures and keeps the maximum temperature below 40°C. Again, this comes at the expense of the performance. Figure 4 shows that the use of 450 MHz and 510 MHz drops almost to zero. Furthermore, the operation at 390 MHz drops from 67% to 51%. In contrast, the operation at the lowest (180 MHz) and the second lowest frequency (305 MHz) increase from 12% to 31% and 0% to 9%, respectively. Consequently, operating at lower frequencies leads to a 32% drop in the median *FPS*, as shown in Table I.

Amazon shopping: Amazon is a popular online shopping app available on the Google play store. In contrast to the gaming apps, it primarily uses the CPU when it is active. Figure 5 shows the behavior of the temperature of the phone when using the Amazon app. We observe that the temperature with and without throttling closely follow each other during the first 80 seconds of use. After that, the temperature increases if the thermal throttling is disabled. In contrast, the thermal governor is able to maintain the temperature by reducing the CPU frequency. The reduction in CPU frequency can be analyzed using the data in Figure 6. When there is no throttling, the CPU operates 32% of the time at 960 MHz. This percentage drops to 23% with throttling. In contrast, the operating at the lowest frequency (384 MHz) increases from 25% to 37% with
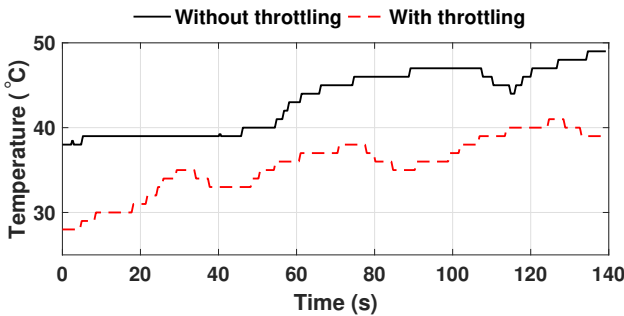

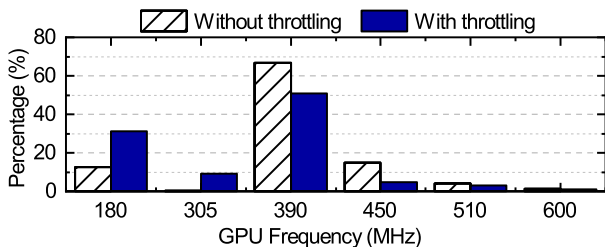Fig. 3: Temperature profile for Stickman Hook game.

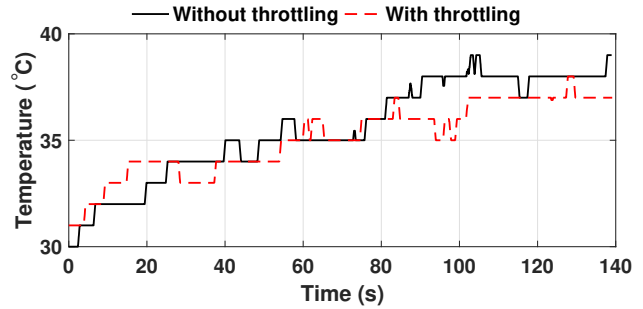Fig. 4: Usage of GPU frequencies in the Stickman Hook game.

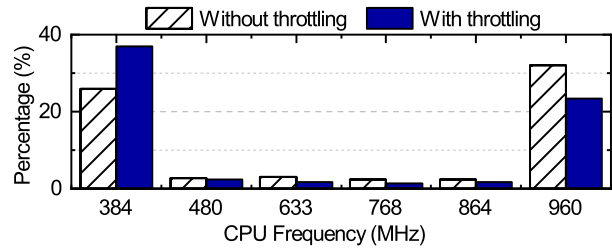Fig. 5: Temperature profile for Amazon shopping app.

Fig. 6: Usage of big core frequencies in the Amazon app.

throttling. As a result, the frame processing rate drops from 35 *FPS* to 28 *FPS* similar to the other applications, as shown in Table I.

Google Hangouts and Facebook apps: In addition to the previous apps, we also analyze the performance of Google Hangouts and Facebook apps. We do not plot the temperature profiles for these two apps since they are similar to that of the Amazon app.

Consistent with the other apps, we observe that the default governor reduces the temperature at the expense of the frame processing rate. In particular, thermal throttling reduces the frame rate from 42 *FPS* to 38 *FPS* while running Google Hangouts. Similarly, the frame of Facebook drops from 35 *FPS* to 24 *FPS* while playing a game in the app.

The experiments with popular Android apps show that thermal governors regulate the temperature at the cost of performance since they *react* to temperature violations by throttling the frequency of all the resources in the system. Theoretical analysis of power-temperature dynamics can help in predicting potential violations before they happen [1, 4, 22]. Furthermore, they can guide the utilization of different resources, such as big versus little cores, judiciously to prevent temperature violations with minimal impact on performance. The next section illustrates this idea with a graphical analysis and empirical results.

IV. THERMAL MANAGEMENT USING POWER-TEMPERATURE STABILITY ANALYSIS

A. Analysis of the Power-Temperature Dynamics

Power and temperature form a well-known positive feedback system [2, 15]. The junction temperature increases with power consumption. Higher temperature, in turn, leads to a higher leakage power, which contributes to a further increase in the temperature [3]. The temperature converges to a stable

fixed point when the system is stable. In contrast, an unstable system experiences a thermal runaway. The stability of power-temperature dynamics depends on the power consumption that changes at runtime. Therefore, it is imperative to analyze the stability of the dynamics at runtime.

A theoretical analysis of the power-temperature dynamics is presented in [2]. This analysis enables us to derive the sufficient and necessary conditions for the existence of temperature fixed points. The fixed points are the *roots* of a concave function of an auxiliary temperature, as illustrated in Figure 7. The auxiliary temperature is inversely proportional to the actual temperature in Kelvin. Therefore, a higher auxiliary temperature corresponds to a lower temperature and vice versa. This section presents a graphical analysis of the stability, while the complete theoretical proof is presented in [2].

Figure 7a plots the fixed-point function when the power consumption is 2 W using parameters obtained for Odroid XU3. We observe that the fixed-point function is indeed concave. Furthermore, it has two roots that correspond to two temperature fixed points. The arrows along the plot show that the auxiliary temperature *decreases* with each fixed-point iteration when the function is *negative*. In contrast, it *increases* at each iteration when the function is *positive* (i.e., between two roots). Hence, the larger root attracts the temperature trajectories to itself, i.e., it is the stable fixed point, as illustrated in Figure 7a. In contrast, the temperature diverges from the unstable fixed point. Hence, any trajectory that starts between the roots will converge to the stable fixed point. If the initial point is to the left of the unstable fixed point, there is a thermal runaway.

When the power consumption increases, the fixed-point function moves down, as shown in Figure 7b. The power-temperature dynamics continue to have two fixed-points until the power consumption reaches a critical value. In our example, the roots of the fixed-point function converge, i.e., there is only one root, when the power consumption reaches 5.5 W. Any further increase in the power consumption results in an unstable system with no fixed points. For instance, Figure 7c plots the fixed-point function when the total power is 8 W. We see that it does not intersect the x-axis, which shows that

the system does not have any fixed points. In summary, we can determine the stability of the power-temperature dynamics by looking at the number of roots of the fixed-point function. In the next section, we use the thermal stability analysis to design an application-aware thermal management algorithm.

B. Application-Aware Thermal Management

The thermal stability analysis overviewed in the previous section provides an efficient method to calculate the steady-state temperature (fixed-point temperature) of the system as a function of the power consumption. We also use it to estimate the time that will pass to reach the fixed point. This information can be used by a governor to enable a new class of dynamic thermal and power management (DTPM) algorithms. As illustrated in Section III, thermal governors in current mobile platforms typically throttle the frequency of all active cores when a thermal violation is detected. Instead of degrading the performance of all active applications in the system, we utilize the fixed-point predictions to design an application-aware thermal management algorithm.

We start by using the thermal stability analysis to determine the stable fixed-point temperature. If this temperature exceeds a specified thermal limit, there may be a thermal violation in the future. Therefore, the algorithm estimates the time it will take for the system to reach the fixed point. If the algorithm detects that the time to reach the fixed-point is less than a user-defined limit, it means that there is an imminent possibility of a thermal violation. In this case, the algorithm finds the process that has the highest power consumption by monitoring the average utilization of each active process for a one-second window. We use a window to filter out momentary peaks in the power consumption. Finally, the algorithm moves the most power-hungry process to low power processors. We repeat this process every 100 ms to capture runtime variations in the system activity effectively.

The primary advantage of the proposed algorithm is penalizing only the process that causes a thermal violation. The other processes continue operating at maximum performance in strong contrast to the default governors, which throttle the whole system to regulate the temperature. The algorithm also lets processes with real-time requirements register themselves
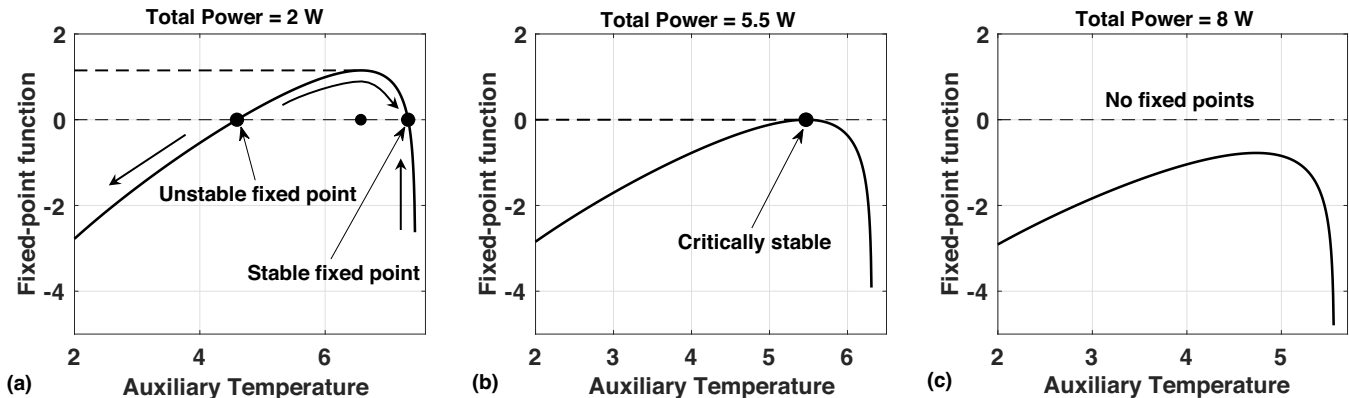

Fig. 7: Illustration of the fixed point functions for three power consumption values.

so that they are not penalized. We present an empirical evaluation of the proposed algorithm in the next section.

C. Experimental results on Odroid-XU3

This section evaluates the proposed application-aware thermal management algorithm on the Odroid XU3 board. The board employs the Samsung Exynos 5422 SoC [10], which integrates four Cortex-A15 (big) cores, four Cortex-A7 cores, and a Mali T628 GPU. In addition to the processing elements, the board includes thermal sensors to measure the temperature of each big core and the GPU. It also provides current sensors to measure the power consumption of the little cluster, big cluster, main memory, and the GPU. We run Android 7.1 with Linux kernel 3.10.9. The governor is invoked every 100 ms, along with the default frequency governors. We evaluate the proposed controller with commonly used 3DMark and Nenamark benchmarks. We use the Odroid XU3 board to illustrate the proposed algorithm since it provides a higher level of control to modify the frequency and thermal governors in the system. Furthermore, the Odroid board provides individual power sensors that allow for a better evaluation of the proposed algorithm.

To evaluate the algorithm, we run a real-time GPU benchmark, along with a computationally intensive task in the background. The default policy is to use the thermal management policy in the Linux kernel (3.10.9). Specifically, it uses thermal trip points and ARM intelligent power allocation algorithm to control the temperature [31]. During these experiments, we disable the fan on the board since it is not feasible for mobile platforms.

3D Mark alone with the default governor: First, we run the 3DMark benchmark alone without any background application. This experiment gives an upper bound for the performance. It also gives the baseline temperature profile, which is shown by the blue curve in Figure 8. The corresponding power consumption and its distribution among the major components on the SoC are shown in Figure 9a. We observe that the GPU has the highest power consumption, followed by the big core cluster. This is expected since 3DMark is a GPU-heavy application.
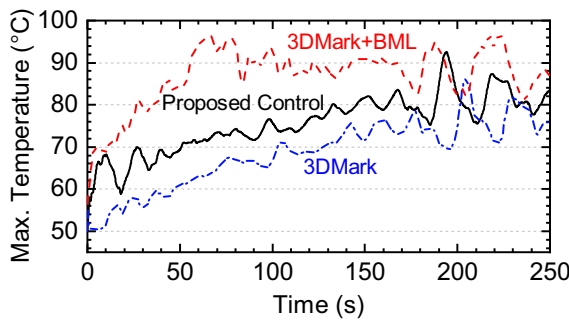

Fig. 8: The maximum temperature of the system when running 3DMark application
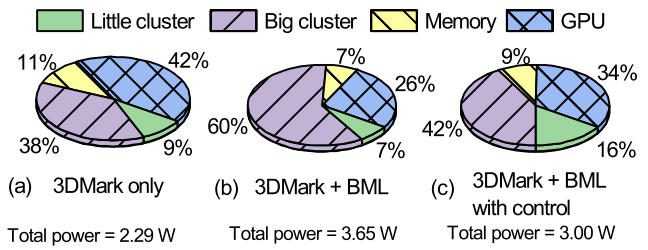

Fig. 9: Power consumption distribution of 3DMark

3D Mark + BML with the default governor: Next, we re-run the 3DMark benchmark while also executing the basicmath large (BML) application [8] in the background. As expected, the BML application increases the power consumption of the big core, and consequently the whole board. More specifically, the total power consumption jumps to 3.65 W, and the contribution of the big core cluster increases from 38% to 60%, as depicted in Figure 9b. Consequently, larger power consumption leads to a higher temperature, as shown by the red dashed line in Figure 8. As a result, the thermal governor starts to throttle the frequency of the system. Throttling of all the resources causes a drop in the performance of 3DMark Graphics Test 1 (GT1) from 97 to 86 *FPS*, as shown in the third column of Table II. Similarly, the performance of Graphics Test 2 (GT2) also decreases from 51 *FPS* to 49 *FPS*.

3D Mark + BML with the proposed controller: Finally, we apply the proposed control algorithm when running 3D Mark and BML together. As in the previous experiment, the BML application causes the temperature to rise. However, the algorithm detects and migrates the background application to the little cluster as soon as it predicts a thermal violation. This enables the proposed algorithm to control the temperature of the system, as shown using a black line in Figure 8. The migration also causes a reduction in the big core power consumption from 60% to 42%, as seen by the pie chart in Figure 9c. We see a corresponding increase in the power consumption of the little core from 7% to 16% since the BML application is running on the little cores. The migration is able to effectively throttle the BML application without affecting the 3DMark application. This can be observed in the last column of Table II, where the performance loss in 3DMark GT1 and GT2 are minimal.

The same steps are repeated with the Nenamark benchmark in order to test the controller on a workload with different

TABLE II: Comparison of application performance with the proposed control algorithm

Test	App. Alone	App. + BML	App. + BML with Proposed Control
3DMark GT1	97 *FPS*	86 *FPS*	93 *FPS*
3DMark GT2	51 *FPS*	49 *FPS*	51 *FPS*
Nenamark3	3.5 levels	3.4 levels	3.5 levels

Design, Automation And Test in Europe (DATE 2019)

characteristics. Nenamark measures the number of levels of the benchmark that can be run at a given frame rate. The benchmark terminates when the frame rate drops below the desired level. The performance achived while running Nenamark under the three scenarios is summarized in the last row of Table II. We observe that the number of levels which can run at the desired frame rate is lower when the whole system is throttled. However, with the proposed control the number of levels is equal to the baseline performance of the application without any background applications. In summary, the proposed control algorithm is able to effectively migrate the power-hungry app without causing any adverse effect on the main app.

V. CONCLUSIONS

This paper presented experiments to study the power, performance and thermal behavior of modern smartphones. Specifically, we performed experiments on the Nexus 6P phone with popular apps to understand the loss in their performance due to thermal throttling. Using insights from these experiments and our previous work, we developed an algorithm that can selectively throttle background apps without affecting foreground apps. Experiments on the Odroid-XU3 board show that the algorithm can throttle background apps effectively. The experimental case study in this paper can be used as a baseline when evaluating future thermal management algorithms. Furthermore, it can be used by application developers to optimize their apps such that they do not experience thermal throttling.

REFERENCES

[1] G. Bhat *et al.*, "Algorithmic Optimization of Thermal and Power Management for Heterogeneous Mobile Platforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 544–557, 2018.

[2] G. Bhat, S. Gumussoy, and U. Y. Ogras, "Power-Temperature Stability and Safety Analysis for Multiprocessor Systems," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 145:1–145:19, 2017.

[3] D. Brooks, R. P. Dick, R. Joseph, and L. Shang, "Power, Thermal, and Reliability Modeling in Nanometer-Scale Microprocessors," *IEEE Micro*, vol. 27, no. 3, pp. 49–62, 2007.

[4] R. Cochran and S. Reda, "Thermal Prediction and Adaptive Control Through Workload Phase Detection," *ACM Trans. Des. Autom. of Electron. Syst.*, vol. 18, no. 1, pp. 7:1–7:19, 2013.

[5] B. Egilmez, G. Memik, S. Ogrenci-Memik, and O. Ergin, "User-Specific Skin Temperature-Aware DVFS for Smartphones," in *Proc. Conf. on Design, Autom. and Test in Europe*, 2015, pp. 1217–1220.

[6] U. Gupta *et al.*, "Dynamic Power Budgeting for Mobile Systems Running Graphics Workloads," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 1, 2017.

[7] U. Gupta, S. Korrapati, N. Matturu, and U. Y. Ogras, "A generic energy optimization framework for heterogeneous platforms using scaling models," *Microprocessors and Microsystems*, vol. 40, pp. 74–87, 2016.

[8] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A Free, Commercially Representative Embedded Benchmark Suite," in *Proc. Int. Workshop on Workload Char.*, 2001, pp. 3–14.

[9] V. Hanumaiah *et al.*, "STEAM: A Smart Temperature and Energy Aware Multicore Controller," *ACM Trans. on Embedd. Comput. Syst.*, vol. 13, no. 5s, p. 151, 2014.

[10] Hardkernel, "ODROID-XU3," https://wiki.odroid.com/old_product/odroid-xu3/odroid-xu3 Accessed 24 Nov. 2018, 2014.

[11] J. Henkel, H. Khdr, and M. Rapp, "Smart Thermal Management for Heterogeneous Multicores," in *Proc. Conf. on Design, Autom. and Test in Europe*, March 2019.

[12] Huawei. Huawei Nexus 6P Smartphone. https://consumer.huawei.com/my/phones/nexus-6p/specs/, Accessed 12/5/2018.

[13] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance For a Given Power Budget," in *Proc. Int. Symp. on Microarchitecture*, 2006, pp. 347–358.

[14] B. K. Joardar, R. G. Kim, J. R. Doppa, and P. P. Pande, "Design and Optimization of Heterogeneous Manycore Systems enabled by Emerging Interconnect Technologies: Promises and Challenges," in *Proc. Conf. on Design, Autom. and Test in Europe*, March 2019.

[15] W. Liao and L. He, "Coupled Power and Thermal Simulation with Active Cooling," in *Int. Workshop on Power-Aware Comput. Syst.*, 2003, pp. 148–163.

[16] A. Miyoshi *et al.*, "Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling," in *Proc. Int. Conf. on Supercomput.*, 2002, pp. 35–44.

[17] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin, "Hierarchical Power Management for Asymmetric Multi-Core in Dark Silicon Era," in *Proc. of DAC*, 2013, p. 174.

[18] National Instruments, "PXIe-4081," http://www.ni.com/en-us/support/model.pxie-4081.html, 2017.

[19] S. Pagani *et al.*, "TSP: Thermal Safe Power: Efficient Power Budgeting for Many-Core Systems in Dark Silicon," in *Proc. Int. Conf. on Hardware/Software Codesign and Syst. Synth.*, 2014, p. 10.

[20] J. Park, S. Lee, and H. Cha, "Accurate Prediction of Smartphones' Skin Temperature by Considering Exothermic Components," in *Proc. Conf. on Design, Autom. and Test in Europe*, 2018, pp. 1500–1503.

[21] F. Paterna, J. Zanotelli, and T. S. Rosing, "Ambient Variation-Tolerant and Inter Components Aware Thermal Management for Mobile System on Chips," in *Proc. Conf. on Design, Autom. and Test in Europe*, 2014, pp. 1–6.

[22] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, "Improving Mobile Gaming Performance Through Cooperative CPU-GPU Thermal Management," in *Proc. of the Design Autom. Conf.*, 2016, pp. 47:1–47:6.

[23] Qualcomm Inc. Qualcomm Snapdragon 810 Processor. https://www.qualcomm.com/products/snapdragon/processors/810.

[24] A. Raghavan *et al.*, "Computational Sprinting," in *Int. Symp. High Perf. Comput. Arch.*, 2012, pp. 1–12.

[25] O. Sahin and A. K. Coskun, "On the Impacts of Greedy Thermal Management in Mobile Devices," *IEEE Embedded Systems Letters*, vol. 7, no. 2, pp. 55–58, 2015.

[26] O. Sahin and A. K. Coskun, "QScale: Thermally-Efficient QoS Management on Heterogeneous Mobile Platforms," in *Proc. Int. Conf. on Comput.-Aided Design*, 2016, pp. 125:1–125:8.

[27] O. Sahin, L. Thiele, and A. K. Coskun, "MAESTRO: Autonomous QoS Management for Mobile Applications under Thermal Constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2018.

[28] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The EDA Challenges in the Dark Silicon Era: Temperature, Reliability, and Variability Perspectives," in *Proc. Des. Autom. Conf.*, 2014, pp. 1–6.

[29] Statista, "Most Popular Google Play App Store Categories From 1st Quarter 2016 to 3rd Quarter 2018, by Number of Downloads (In Millions)," https://www.statista.com/statistics/256772/most-popular-app-categories-in-the-google-play-store/ Accessed 4 Dec. 2018.

[30] S. Wang, G. Ananthanarayanan, and T. Mitra, "OPTiC: Optimizing Collaborative CPU-GPU Computing on Mobile Devices with Thermal Constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2018.

[31] X. Wang. (2015) Intelligent Power Allocation. http://infocenter.arm.com/help/topic/com.arm.doc.dto0052a/DTO0052A_intelligent_power_allocation_white_paper.pdf, Accessed 4 Dec.2018.