Ambient variation-tolerant and inter components aware thermal management for mobile system on chips

Francesco Paterna[†], Joe Zanotelli[‡] and Tajana Simunic Rosing[†]
[†]Dept. of Computer Science and Engineering University of California, San Diego email:{fpaterna|tajana}@eng.ucsd.edu
[‡]Qualcomm Technologies, Inc. email:josephz@qti.qualcomm.com

Abstract— **In this work we measure and study two key aspects of the thermal behavior of smartphones: 1) thermal interaction between the components on the printed circuit board and 2) the influence of phone's ambient temperature which is subject to large variations. The measurements on the smartphone running typical workloads show that the heat generated by the communication subsystem and the high temperatures on the back cover of the phone can increase the SoC temperature by as much as 17^{o}C. None of the run-time thermal management studies presented to date considered this interaction, as there was no model available. We design a thermal model that captures this thermal dependency and a policy able to avoid thermal emergencies while minimizing the impact on performance.**

I. INTRODUCTION AND RELATED WORK

Smartphones are composed by several ICs components (e.g. SoC, DRAM, RX/TX) and devices (e.g. camera, display) which work simultaneously and generate heat. The absence of active cooling devices along large ambient condition variations creates a complex thermal behavior. More specifically, significant thermal interactions occur between SoC, communication module, audio codec, DC-DC converter, battery and display as a function of the changing ambient conditions which are consistent on the back cover of the phone.

Recent publications model the thermal behavior of the multiprocessor systems on chips (MPSoCs) while assuming a stable ambient temperature next to it and disregarding the rest of the system [1][2][4][5][6][7][8]. A few studies to date attempted to go beyond just SoC thermal modeling. In [5] a smartphone and a tablet are disassembled and the thermal properties, such as thermal conductivity and heat capacity of the package including cover, display and battery are measured. The thermal study of the entire working phone has not been conducted. In [6], power and energy analysis of the different components of a smartphone such as CPU, memory and display under different usage conditions such as voice call and message texting, is performed. Similar analysis is done in [3]. Here the authors also take into account the power dissipated by the voltage regulator and DC-DC converter. Their main goal is to provide an infrastructure to evaluate the battery lifetime.

In this paper we first analyze the effect of the ambient condition variations and inter-component dependency on the temperature of the SoC and then we build an efficient and accurate thermal model. We finally use our model to devise a policy aware of the undesired effects on the SoC to adjust the clock frequency with the aim of meeting thermal constraints while minimizing the impact on performance. In our experiments, we adopt one of the recent commercial smartphones and use an industrial profiler to collect performance and thermal statistics. We also used thermistors connected to a digital analyzer to measure the temperature on the back cover of the phone under different conditions such as phone lying on a desk and phone held in a hand. We use a set of benchmarks characterized by 2D/3D rendering and arithmetic computations to exercise the SoC. As inter-component dependency, we focus on the RX/TX since it results a significant heat source along with the SoC. The contribution of this paper is threefold and can be summarized as follows.

As **first contribution**, we show through experiments on our target phone that the combined effect of ambient condition variations and thermal interaction between SoC and RX/TX can increase the SoC temperature by as much as +17^{o}C. If thermal increase is avoided by slowing down computing, the resulting performance loss is as much as 27%.

As **second contribution**, we propose a thermal model of the smartphone systems. Our model is composed by two sets of functions. The first one is able to estimate the thermal increment on the back cover of the phone over time in two different conditions: phone lying on a desk and phone held in hand. We suppose to be able to catch the changing of the condition by exploiting the face detector sensor mounted on the front cover of the phone: if the user is looking at the screen, it is more likely he is holding the phone in his hand. The other set of functions is able to predict the SoC temperature after a period as long as 10 sec by exploiting the estimated back cover temperature and the knowledge about the RX/TX activity, the clock frequency and the current SoC temperature read by the sensor. We validate our model against real measurements obtained from our target phone over different contexts in terms of ambient conditions and RX/TX activity. Our results show that the average error of the model is below 1^{o}C.

As **third contribution** we devise an ambient variation-tolerant and inter components aware thermal management technique able to meet thermal constraints set up for the SoC while minimizing the impact on performance. We prove through experiments on our target phone that our approach has adaptive properties with respect to the ambient condition variations and the inter-component dependency.

The rest of this paper is organized as follows. In Section II we detail our experimental setup. In Section III we evaluate the impact of the ambient condition variations and the inter-component dependency on the SoC temperature. In Section IV we present our thermal model and our policy. In Section V we provide experimental results about the model

978-3-9815370-2-4/DATE14/©2014 EDAA

validation and the policy effectiveness. Section VI concludes this paper.

II. Experimental setup

This paper's key contributions hinge on experiments performed on a smartphone to show and quantify the effects of the ambient conditions and the activity of the RX/TX module on the SoC performance. Thus, we start by describing the phone we used and the experimental setup.

A. Target phone characteristics

Our target phone is Snapdragon^{TM} smartphone by Qualcomm® [11]. Its printed circuit board (PCB) has components on both sides. On the top of the PCB there are the MSM8660 SoC and the Qualcomm® QTR8615 RX/TX communication module. The key components on the SoC are two ARM15[13] CPUs and Adreno^{TM} GPU[12].

We collect detailed power, performance and thermal data using the Trepn profiler [10]. Profiler's sampling resolution is 100ms. We collected power measurements of the SoC's two ARM15 application CPUs, a GPU and DRAM as well as the clock frequency of the CPUs and the GPU. We set the phone in the airplane mode and activate the WiFi which has always been connected to our network. The phone is first placed on a desk during the first phase of experiments and then is handheld. We also performed another set of experiments by placing the back of the phone 10 cm above a 400W lamp to increase the temperature surrounding the phone and hence mimic what happens if the phone is handheld over long periods. We measure the temperature on the back of the phone by using two thermistors [14] and the analyzer USB-6216 by National Instruments [15]. We capture the traces by adopting LabView [16]. The locations on the back cover of the two thermistors were in correspondence of the SoC and the RX/TX respectively as these components are mounted on the PCB inside the phone. We average the values provided by the two thermistors.

B. Benchmarks

As benchmark for the experiments, we use the 2D and 3D rendering samples and the Math samples found within the 0xBenchmark [17] suite. The 2D set is composed by 7 benchmarks detailed as follows.

1. **Canvas Redraw**. It uses random color to redraw canvas repeatedly.
2. **Draw Circle**. A simple 2D animation program.
3. **DrawRect**. It repeatedly adds random colored, size rectangles on canvas.
4. **DrawCircle2**. It repeatedly renders random colored, size circles on canvas.
5. **DrawArc**. A Simple 2D animation.
6. **DrawText**. It calculates text rendering speed.
7. **DrawImage**. It calculates picture rendering speed.

The 3D set is composed by 4 benchmarks detailed as follows.

1. **GL Cube**. A sample program from the Android API Demo that uses OpenGL ES to render a rotating Rubik's Cube.
2. **GL Teapot**. It uses OpenGL ES to render a rotating Utah Teapot.
3. **NeHe Lesson08**. A rotating 3D cube with textured applied and alpha blending enabled.
4. **NeHe Lesson16**. A rotating 3D cube with textured applied and GLFog enabled.

The Math set is composed by 7 benchmarks detailed as follows.

1. **Linpack**. It makes use of a general numerical linear algebra operation to test the device's ability to perform floating point operations (MFLOPS).
2. **Scimark2**. It includes six scientific calculations (Composite, FTT, SOR, MonteCarlo, SparseMatMult and LU) for testing the device's abilitity to perform floating point operations (MFLOPS).

We ran these three sets of benchmarks in different conditions with respect to the temperature on the back of the phone and the WiFi activity. To exercises the WiFi, we launched a downloading from Gmail of a 400Mbyte file via the ONE Browser app [18]. We used this size to make sure that the downloading lasts as long as the benchmark executions. We were not able to activate the 3g in our phone.

III. Thermal effects on the smartphone system

In this section, we analyze through real measurements the impact of the ambient condition variations and the WiFi activity on the SoC temperature and the consequent performance loss.

In the **first experiment**, we use the rendering benchmarks. We have a 2D and a 3D set of execution samples. The 2D set takes roughly 2 min while the 3D 2.5 min. Each experiment is characterized by the execution of the 2D or the 3D set multiple times in sequence. Each experiment is repeated four times in according to four different contexts specified as follows. 1) Back cover temperature 30^{o}C; 2) Back cover temperature 30^{o}C while WiFi is downloading; 3) Back cover temperature 45^{o}C; 4) Back cover temperature 45^{o}C while WiFi is downloading. Please note 45^{o}C is usually the thermal constraint on the back cover of the phone since it is the maximum temperature the human skin can tolerate [21]. We report the SoC temperature and the performance evaluated in frame-per-second in Tables I,II in which each row refers to the execution of one 2D/3D set over the sequence. For both the 2D and the 3D analysis, the WiFi makes the SoC temperature increase up to 8^{o}C and the performance drop by as much as 22%. When the back cover temperature reaches 45^{o}C, the SoC temperature increases by as much as 17^{o}C.

T (ºC)	Draw Canvas	Draw Circle	Draw Circle2	Draw Rect	Draw Arc	Draw Image	Draw Text
38	Back temperature 30^{O}C						
41	60.29 fps	67.11	57.60	49.21	57.79	59.86	57.57
42	60.38 fps	67.94	58.18	51.35	58.08	60.16	59.62
43	60.34 fps	67.00	58.29	50.14	57.77	59.84	59.48
44	60.43 fps	65.50	58.81	49.27	57.90	59.42	58.81
44	60.40 fps	67.05	58.67	50.04	57.51	59.85	59.87
38	Back Temperature 30^{O}C + WiFi downloading						
47	60.86 fps	53.68	58.22	42.21	51.26	54.10	56.93
48	60.71 fps	57.52	58.72	39.91	51.20	51.22	55.91
51	60.67 fps	57.19	57.57	42.39	52.64	50.53	52.33
51	60.52 fps	52.93	59.23	44.80	48.43	54.51	52.91
48	60.45 fps	65.74	58.55	51.16	57.38	59.48	59.26
47	60.86 fps	53.68	58.22	42.21	51.26	54.10	56.93
38	Back temperature 45^{O}C						
45	60.37 fps	67.7	58.27	50.36	58.92	59.73	59.26
50	59.90 fps	68.16	57.69	52.27	58.31	59.72	59.61
51	60.62 fps	66.98	58.9	51.33	58.17	59.89	59.69
54	60.41 fps	67.12	58.72	51.77	58.65	59.69	59.47
53	60.54 fps	68.31	58.72	51.45	58.55	60.18	59.52
38	Back temperature 45^{O}C + WiFi downloading						
54	60.93 fps	56.23	60.31	41.35	48.19	44.49	48.7
59	60.84 fps	58.21	60.29	42.66	48.94	48.51	49.95
57	60.76 fps	60.20	60.27	43.98	49.69	52.54	51.21
59	60.84 fps	58.99	58.15	42.20	50.11	49.19	48.88
59	60.97 fps	58.48	59.55	47.50	47.78	49.26	50.67

Table I SoC temperature while running a sequence of the 2D suite. Performance is reported for each benchmark by frame-per-second (fps).

For the **second experiment**, we use the Math suite. We repeatedly execute the set of computational samples for 13 times. Each experiment has been done for three frequencies `{1180MHz, 864, 540}` for the four different contexts described in the first experiment.

We adjust the clock frequency of the CPUs by using the Android Tuner app [20]. We capture the SoC temperature at 0.1sec granularity. Figure 1 shows the temperature over time. First row: back cover is 30ºC. Second row: cover is 45ºC. First column: no downloading. Second column: downloading. When downloading the maximum SoC temperature increases up to 9ºC. We analyzed these data to build a thermal model of the SoC temperature with respect to the back cover temperature, the CPU frequency and the WiFi activity. This study is reported in the next section.

For the **third experiment**, we analyze the back cover temperature increase over time. Figure 2 shows the back temperature increase over 36 minute executions. When the phone is held in a hand, the temperature quickly reaches the human skin temperature 36ºC, after then the increase is linear. We report the trendline which shows that the increment is as much as 0.001ºC/sec. The usage of the WiFi introduces an overall increase by +1ºC. A similar behavior happens when the phone is lying on a desk. The increment is as much as 0.002ºC/sec.

Temp. (ºC)	OpenGL Cube	OpenGL Blending	OpenGL Fog	Flying Teapot
38	Back temperature 30^{O}C			
43	60.04	64.24	64.28	60.60
43	60.08	64.18	64.09	60.59
44	60.11	64.12	63.87	60.60
45	60.11	64.22	64.23	60.60
38	Back Temperature 30^{O}C + WiFi downloading			
47	60.14	64.04	63.80	49.44
52	60.08	64.28	63.67	47.61
53	60.05	63.79	63.85	49.44
54	60.12	64.12	64.00	51.58
38	Back temperature 45^{O}C			
51	59.97	60.13	60.15	60.12
55	64.29	64.09	64.12	64.16
59	63.92	64.15	64.25	64.11
59	53.41	55.55	55.54	55.55
38	Back temperature 45^{O}C + WiFi downloading			
57	60.11	60.12	60.1	60.14
61	64.1	63.94	64.28	63.95
55	63.96	64.11	63.99	64.13
58	51.27	51.28	49.45	49.45

Table II SoC temperature while running a sequence of the 3D suite. Performance is reported for each benchmark by frame-per-second (fps).

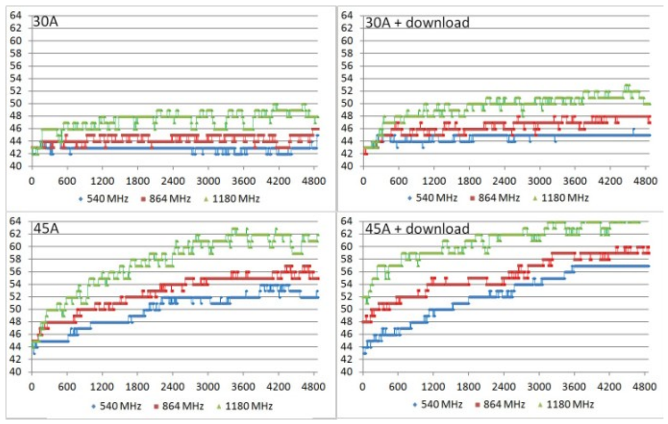

Figure 1 Execution of the Math suite for 13 times in a row. Plots show the SoC temperature over time. First row: back cover is 30ºC. Second row: cover is 45ºC. First column: no downloading. Second column: downloading.

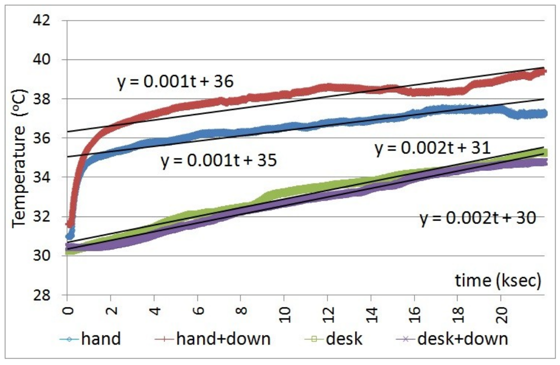

Figure 2 Analysis of the back cover temperature variation over time for different contexts. Linear trendlines reported: y(ºC) = m·t(sec) + q.

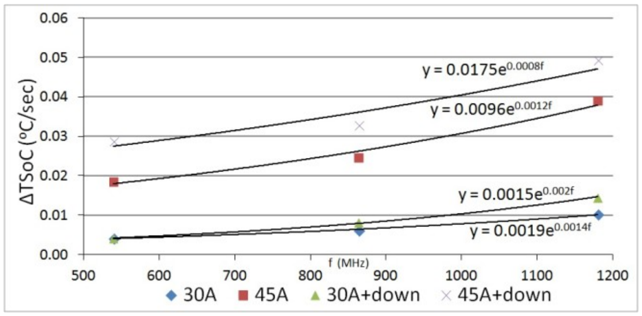

Figure 3 Analysis of the SoC thermal gradient per second for different contexts in terms of back cover temperature and WiFi usage. Exponential trendlines reported: y[ºC/sec] = l·exp(h·f[MHz]).

IV. Thermal Model and Management

A. Model

We define our thermal model as a function able to estimate the SoC temperature after a fixed time by the knowledge about the current SoC temperature supposed read by a sensor, the CPU frequency, the temperature on the back cover and the WiFi activity (i.e. whether it is downloading). In particular the model is able to estimate how the thermal increment per second changes with respect to the factors we mentioned above. The model we propose is based on the observations related to the analysis in the previous section. We also validate its accuracy in the results of the next section.

Since in the practical usage, we are not able to measure the temperature on the back cover, we need a model also for it. We based on the third experiment presented above. The back temperature changes very slowly with respect to the SoC temperature and we can take into account this behavior by assuming a linear trend as this well shapes the curves for the all four cases of Figure 2. We can discriminate between phone handheld in which the increment is as much as 0.001ºC/sec and phone lying on desk in which the increment is as much as 0.002ºC/sec. To know whether the user is holding the phone in his hand, we can base of what the face detector on the front cover captures; in particular if the user is looking at the phone, probably the phone is handheld. Also, when the ambient condition changes we can add a ±5ºC in according to whether the phone passes from the desk in a hand or vice versa. Hence, we define the model for the increment of the back cover temperature ΔT_a in (1) in which t is the time in sec elapsed from the previous observation.

$$\Delta T_a = \begin{cases} 0.001 \cdot t & \text{hand } (+5 \text{ when switch}) \\ 0.002 \cdot t & \text{desk } (-5 \text{ when switch}) \end{cases} \qquad (1)$$

We can now proceed with the model of the SoC temperature with respect to the CPU frequency. We do not model the GPU since we are not able to change the GPU frequency in our phone. As in the second experiment of the previous section, we collected thermal traces for the SoC at

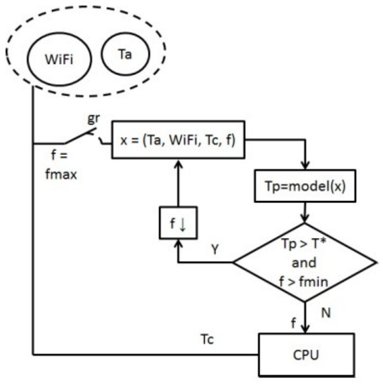

Figure 4 Proposed algorithm diagram.

two different back cover temperatures and two different types of WiFi activities for three different CPU frequencies. We report in Figure 3 the four different trends over the CPU frequencies. For the all four cases which differ in terms of WiFi activity and back cover temperature, the trend over the clock frequency can well approximated by exponential curves. The idea could be to understand how the parameters of the exponential function depend from the back cover temperature and the WiFi activity in terms of data transferring type (i.e. download/upload) and signal. We are not able to act on the data transferring rate and signal, hence we only discriminate whether the WiFi is downloading or not. Also we did not consider the upload as we assume that effect is comparable to the download. For each of the four cases, we associate a different function to model the SoC temperature expressed by `y[°C/sec] = l·exp(h·f[MHz])`. We assume that the coefficient l changes linearly with respect to thermal gradient on the back cover. Instead, we assume that the h coefficient is constant when the WiFi is not working; we average the values obtained at the two back cover temperatures. When the WiFi is downloading we also assume a linear variation of the coefficient h with respect to the thermal gradient on the back cover. The resulting functions are reported in Equation (2). The final SoC temperature is calculated through Equation (3), in which the SoC thermal increment calculated through (2) is multiplied by the slot time Δt and then added to the temperature read by the sensor.

$$\Delta T_{SoC}(t) = \begin{cases} \left(19 + 5 \cdot \Delta T_a\right) \cdot 10^{-4} \exp\left(13 \cdot 10^{-4} \cdot f\right) \\ \text{no downloading} \\ \left(15 + 11 \cdot \Delta T_a\right) \cdot 10^{-4} \exp\left(\left(20 - 0.8 \cdot \Delta T_a\right) \cdot 10^{-4} \cdot f\right) \\ \text{downloading.} \end{cases} \qquad (2)$$

$$T_p(t + \Delta t) = \Delta T_{SoC}(t) \cdot \Delta t + T_c(t) \qquad (3)$$

B. Policy

We leverage the models we developed to design a policy that improves performance of the CPU as a function of ambient conditions and RX/TX activity without causing violations of SoC thermal constraints. Our policy is designed for the Math benchmark suite illustrated above. This set of benchmark results no memory bound which means that the execution time of each singular samples linearly increases with respect to the frequency. For instance for Linpack the Mflops are {14.28, 30. 61, 48.24} at {540, 846, 1180 MHz} respectively. We do not report the same information for the all suite for space limitation. In addition the power linearly scales with respect to the frequency too; for the same frequency group the average power results {0.10, 0.17, 0.23 W} respectively. We conclude that 1) from the energy point of view there is no preferred frequency 2) from the performance point of view the higher the better 3) from the thermal point of view a maximum allowed frequency may exist and may vary over time. We then devise a technique which increases as much as possible the CPU frequency over time by avoiding thermal constraint violations. Before giving details about the policy we define the adopted variables as follows.

- ***f***: CPU frequency (**fmax:** max freq. - **fmin** min freq.);
- ***Tp***: predicted SoC temperature;
- ***WiFi***: a Boolean value to specify whether the WiFi is downloading;
- ***Ta***: back cover temperature;
- ***Tc***: actual SoC temperature read by the sensor;
- **gr**: time granularity of the time slot Δt;
- ***T****: SoC temperature constraint;
- ***model()*** : function implementing the thermal model.

Figure 4 illustrates the proposed algorithm. Each period of time *gr*, the current back cover temperature is estimated through (1), the WiFi activity is sampled, the SoC temperature is read from the sensor. The frequency which is the output of the algorithm is set to its maximum value. All this information composes the variable *x* which is used to estimate the SoC temperature after the time *gr* through (2) and (3). The predicted temperature is compared with the thermal constraint and as long as that value results larger, the frequency is decreased by one step in according to the possible settings. If the predicted temperature is smaller than the threshold or the frequency has reached the minimum possible value, the algorithm exits and the frequency is set to the CPU.

V. Results

A. Model validation

In this first part of results we validate our thermal model presented above. To prove that our model well captures ambient variations and inter component interactions we execute the Math suite continuously for 7500sec. During this time we changed ambient condition by holding the phone in a hand or letting it lying on a desk. We also download a file for some periods and change the CPU frequency. How we divided the time slots is specified in Figure 5. We collect our thermal traces related to the SoC and the back temperature with a granularity by 0.1sec. Our model estimates the SoC temperature at the next instant by the knowledge of the current temperature, the back temperature and the WiFi activity (i.e. whether it is downloading). When calculating the thermal gradient on the back cover, both the values are estimated by using Equation (1), except for the first value. We validate our model by projecting the temperature at three different granularities: 1sec, 5sec, 10sec. Figure 5 shows that the model results accurate and sensitive to the all variations in terms of CPU frequency, ambient conditions and WiFi activity. The average error is less than 1^{o}C and the maximum is below 4^{o}C; a summary is reported in Table III.

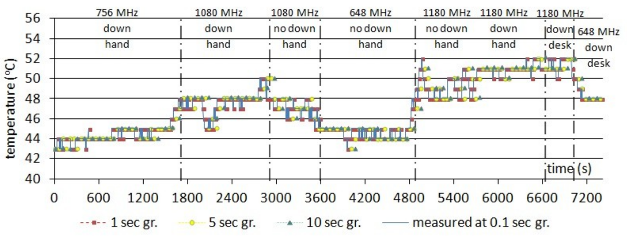

Figure 5 Measured vs. estimated SoC temperature over time.

	1 sec gr.	5 sec gr.	10 sec gr.
Av.	0.22	0.55	0.82
Max	2.01	3.07	3.91

Table III Average and maximum estimated temperature error (^{o}C).

	Back 36 ºC		Back 36 ºC + down.		Back 30 ºC	
time (sec)	f (MHz)	T (^{o}C)	f (MHz)	T (^{o}C)	f (MHz)	T (^{o}C)
t_0	756	45	540	45	756	45
t_0+100	1180	48	1180	52	1180	47
t_0+200	1180	49	540	47	1180	48
t_0+300	918	45	1180	52	1180	50
t_0+400	1180	49	540	48	540	44
t_0+500	918	47	1180	52	1180	49
t_0+600	1180	50	540	48	918	47

Table IV Runtime policy acts to increase the CPU frequency as much as possible while holding the CPU temperature below 50^{o}C. Three different contexts related to the back cover temperature and the WiFi activity.

B. Policy Effectiveness

In this experiment we prove the effectiveness of our strategy in managing the CPU performance in different contexts of ambient conditions and RX/TX activity. We run in our phone the Math benchmark suite and execute in a row all the samples whose details are reported above. We check the SoC temperature each 100 sec through the *adb* shell [19] on our desktop computer. Indeed our phone is connected to our computer via usb. We implemented the policy algorithm and the models we illustrated above in Matlab. The Matlab routines we implemented calculate the new frequency of the CPU which will try to avoid the thermal violation while impacting as less as possible the performance. The routines, as explained in the policy section, also need the information about the temperature on the back cover of the phone and the activity status of the WiFi; i.e. whether it is downloading. We adjust the clock frequency of the CPUs by using the Android Tuner app. The possible clock frequencies are: `{1180MHz, 1130, 1080, 1020, 972, 918, 864, 810, 756, 702, 648, 594, 540}`.

We use three different contexts to prove the two properties of our management technique: 1) ambient-variation tolerance and 2) inter component awareness. In the **first** one the back phone starting temperature is 36^{o}C; our phone was in particular held in a hand. The **second** is as the first but we also activate a file downloading from Gmail by using ONE Benchmark. The file size is 400 Mbyte and we repeatedly launched the downloading to cover the all length of the experiment. In the **third** context, the phone was lying on a desk and the back phone starting temperature is 30^{o}C. The SoC thermal constraint is 50^{o}C.

We start to activate our observations when the SoC temperature is stable to 45^{o}C. Table IV shows, for the different contexts, the frequency set by the policy and the consequent SoC temperature after 100 sec running; the policy then decides the new frequency reported at the row below. We only report the first seven traces since the behavior is actually recurring. For the first context we can see that the frequency is set to the maximum value and when the temperature is 49^{o}C to avoid the thermal violation the frequency is decreased to 918MHz. One time slot later, the high performance can be resumed. When downloading the file the behavior is more critical, any time the policy decides to raise up the frequency to the maximum value the temperature will slightly violate the constraint but then the policy is able to reestablish the thermal safety by decreasing the frequency accurately. In third context, in which the phone is lying on desk and then the back temperature is cooler, our policy is able to capture the possibility to hold the maximum frequency for a longer period without violating the thermal constraint.

VI. Conclusion

In this paper we face the problem of the thermal modeling and management of mobile system on chips. Using an experimental setup composed by a commercial smartphone running typical graphic and computational benchmarks we show that the temperature variation on the back of the phone and the WiFi activity can increase the SoC temperature up to 17^{o}C. If thermal increase is avoided by slowing down computing, the resulting performance loss is as much as 27%. To face this problem we first elaborate a thermal model and then a policy which is able to contrast the ambient variations and the inter-component dependency by acting on the computational speed of the SoC in an efficient way. We validate our model and prove the proposed technique through experiments on our target phone.

VII. Acknowledgements

This work was supported by Qualcomm, SRC (Task ID 2169.001) and NSF variability (g.a. 1029783).

Refereence

[1] E. Ardestani, A. Ziabari, A. Shakouri, and J. Renau. Enabling power density and thermal-aware floorplanning. In *Proceedings of SemiconductorThermal Measurement and Management Symposium*, pages 302–307.IEEE, 2012.

[2] 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 Proceedings of the Design, Automation Test in Europe Conference Exhibition*, pages 1–6. IEEE, 2011.

[3] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In Proceedings of the conference on USENIX annual technical conference, pages 21–21. USENIX Association, 2010.

[4] Y. Ge, P. Malani, and Q. Qiu. Distributed task migration for thermal management in many-core systems. In *Proceedings of the International Conference on Design Automation Conference*, pages 579–584. IEEE, 2010.

[5] S. Gurrum, D. Edwards, T. Marchand-Golder, J. Akiyama, S. Yokoya, J. Drouard, and F. Dahan. Generic thermal analysis for phone and tablet systems. In *Proceedings of the International Conference on Electronic Components and Technology*, pages 1488–1492. IEEE, 2012.

[6] G. P. Perrucci, F. Fitzek, and J. Widmer. Survey on energy consumption entities on the smartphone platform. In *Proceedings of the International Vehicular Technology Conference*, pages 1–6, 2011.

[7] S. Sharifi, R. Ayoub, and T. Rosing. Tempomp: Integrated prediction and management of temperature in heterogeneous mpsocs. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition*, pages 593–598. IEEE, 2012.

[8] J. W. Sheaffer, K. Skadron, and D. P. Luebke. Studying thermal management for graphics-processor architectures. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, pages 54–65. IEEE, 2005.

[9] A. Sridhar, A. Vincenzi, M. Ruggiero, T. Brunschwiler, and D. Atienza. 3d-ice: Fast compact transient thermal modeling for 3d ics with intertier liquid cooling. In *Proceedings of the International Conference Computer-Aided Design*, pages 463–470. IEEE, 2010.

[10] Qualcomm developer network. Trepn. July 2013 https://developer.qualcomm.com/mobile-development/performance-tools/trepn-profiler [online].

[11] Qualcomm developer network. Snapdragon™ S4 Plus MSM8960 MDP. July 2013 https://developer.qualcomm.com/mobile-development/development-devices/snapdragon-s4-msm8960-mdps [online].

[12] Qualcomm developer network. Adreno. July 2013 https://developer.qualcomm.com/discover/chipsets-and-modems/adreno-gpu [online].

[13] ARM Ltd. Cortex A15. July 2013 http://www.arm.com/products/processors/cortex-a/cortex-a15.php [online].

[14] QTI. September 2013. http://www.thermistor.com/ [online].

[15] NI. September 2013 http://www.ni.com/data-acquisition/ [online].

[16] Labview. September 2013. http://www.ni.com/labview [online].

[17] 0xBenchmark. September 2013. http://code.google.com/p/0xbench/ [online].

[18] ONE Browser. September 2013 http://one-browser.com/mobile/home [online]

[19] Android Debug Bridge. September 2013. http://developer.android.com/tools/help/adb.html [online]

[20] Android tuner app. September 2013. https://play.google.com/store/apps/details?id=ccc71.at&hl=en [online]

[21] V. Wienert, H. Sick and J. zur Mühlen. Local thermal stress tolerance of human skin. In *Anasth Intensivther Notfallmed*, 18(2):88–90. 1983.