Stabilizing CPU Frequency and Voltage for Temperature-aware DVFS in Mobile Devices

Jae Min Kim, Young Geun Kim, *Student Member, IEEE,*
and Sung Woo Chung, *Member, IEEE*

Abstract— Recent mobile devices adopt high performance processors to support various functions. As a side effect, higher performance inevitably leads to power density increase, eventually resulting in thermal problems. In order to alleviate the thermal problems, off-the-shelf mobile devices rely on DVFS (Dynamic Voltage-Frequency Scaling) based DTM (Dynamic Thermal Management) schemes. Unfortunately, in the DVFS based DTM schemes, an excessive number of DTM operations worsen not only performance but also power efficiency. In this paper, we propose a temperature-aware DVFS scheme for Android-based mobile devices to optimize power or performance depending on the option. We evaluate our scheme in the off-the-shelf mobile device. Our evaluation results show that our scheme saves energy consumption by 12.7%, on average, when we use the power optimizing option. Our scheme also enhances the performance by 6.3%, on average, by using the performance optimizing scheme, still reducing the energy consumption by 6.7%.

Index Terms—Dynamic Thermal Management, Power Management, Dynamic Voltage and Frequency Scaling, Smartphone

1 Introduction

Earlier mobile devices were equipped with relatively lower performance microprocessors, due to the limitation of size and power. However, with the advance of process technology, mobile processor vendors provide multi-core processors of small size running at more than 1 GHz frequency, which still consumes extremely low power compared to desktop processors. Nevertheless, as the processor size scales down, the power density of the processors increases despite the reduction in total power consumption. The increased power density eventually increases on-chip temperature, which causes the reliability problem including permanent damage on the system in the worst case.

Though the thermal problems have been seriously addressed for the past decade, they are still crucial especially in mobile devices. While desktop and server computers have powerful cooling solutions to cool down their processors, most mobile devices do not have them due to the limited space and power consumption. For instance, some of the high-end computer systems utilize liquid cooling to alleviate the thermal problems. However, it is not feasible to apply liquid cooling to mobile devices - even a fan or heat sink is not applied to current mobile devices. Thus, it is necessary to resolve the thermal problems with software thermal management methods at the OS level.

One of the widely adopted software thermal management techniques is DTM (Dynamic Thermal Management) scheme. The DTM schemes of the off-the-shelf smartphones monitor temperature from on-chip thermal sensors. As soon as the on-chip temperature increases up to the predefined thermal threshold, voltage and frequency are lowered. On the other hand, voltage and frequency are raised when the temperature is decreased. Though such policy prevents the on-chip temperature from rising above the thermal threshold, it also causes frequent scaling of the frequency and voltage. Since the power consumption is proportional to voltage squared and frequency [2], unstabilized voltage and frequency worsen the power efficiency[1] [3][8] – we formulate this based on Jensen's inequality [20] in Section 3.2.1.

Motivated by this fact, we propose a temperature-aware DVFS (Dynamic Voltage and Frequency Scaling) that saves power and enhances performance through frequency stabilization. Note that voltage is also stabilized when operating with the stabilized frequency. For our temperature-aware DVFS, we provide two options: power optimizing option and performance optimizing option. When the power optimizing option is used, our scheme stabilizes the CPU at the lowest frequency that maintains performance same as the conventional DTM. On the other hand, when performance optimizing option is used, our scheme tends to stabilize the CPU at the highest frequency that still ensures reliability. To the best of our knowledge, our paper is the first academic research to propose, implement, and evaluate OS-level thermal management on the off-the-shelf mobile device. The evaluation results based on real measurements show that our scheme reduces system power and enhances performance without violating thermal constraints.

The rest of the paper is organized as follows. We introduce previous studies on thermal management in Section 2. We propose our temperature-aware DVFS scheme in Section 3. Then, we evaluate the proposed scheme in terms of temperature, power, and performance in Section

• *J.M. Kim, Young Geun Kim, and S.W. Chung are with the Department of Computer and Radio Communication Engineering, Korea University, Seoul 136-713, Korea. E-mail: {joist, carrotyone, swchung}@korea.ac.kr*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.

[1] In this paper, performance divided by power is referred as power efficiency.

Digital Object Indentifier 10.1109/TC.2013.188 0018-9340/13/$31.00 © 2013 IEEE

4. Finally, we conclude our work in Section 5.

2 Related Work

Thermal problems have been studied in the various levels [10]. From the mechanical side, many efficient cooling solutions have been proposed, such as air cooling with a faster fan or larger heat sink [5][14] and liquid cooling [4][9]. These cooling solutions reduce the on-chip temperature without performance degradation. However, mechanical cooling system is limitedly adopted, since it has large size or nonnegligible power consumption. Thus, there also have been many researches on the software thermal management techniques.

For the software thermal management, DVFS is most widely used. Hanson et al. investigated and characterized the thermal response of the processor to DVFS [6] on a real system. They investigated that DVFS has an immediate influence on the on-chip temperature. Thus, DVFS can be a viable solution for thermal management. Liu et al. proposed a design time modeling of DVFS [13] to provide thermal optimization, which prevents run-time thermal emergencies while optimizing the cooling cost and performance. Hanumaiah and Vrudhula proposed the temperature-aware DVFS scheme considering the hard real-time applications [7]. Their scheme makes use of accurate power and thermal models to meet the deadlines of the real-time applications while satisfying the temperature constraints.

In addition to design time thermal management, there have also been many studies for run time thermal management [1][11][12][17]. Brooks and Martonosi proposed a DTM scheme for the high performance microprocessors to maintain the reliability with reduced cooling cost [1]. Skadron proposed a hybrid DTM that combines fetch gating and dynamic voltage scaling to minimize the performance penalty [17]. While most DTM schemes rely on on-chip thermal sensors, Lee et al. addressed the thermal problem caused by limited number of the thermal sensors [11]. In order to overcome the problem, they proposed a predictive temperature-aware DVFS scheme that predicts the on-chip temperature based on the performance counter and simple regression model. Liu et al. also proposed a novel task migration strategy using temperature prediction technique [12].

Shin et al. proposed the collaborative use of mechanical cooling solutions and the software thermal management [16]. Their evaluation results show that the DVFS scheme that considers the cooling cost reduces the energy consumption and enhances the reliability at the same time. On the other hand, mobile devices mostly rely on the software thermal management schemes due to the limitation in the size and power budget. Off-the-shelf smartphones with Android OS (Operating System) leverage the DVFS-based DTM schemes for thermal management. However, the DTM schemes used in mobile devices degrade performance, which also leads to power inefficiency, as explained in Section 1. In our paper, we propose a temperature-aware DVFS scheme to improve performance and power efficiency compared to the conventional schemes.

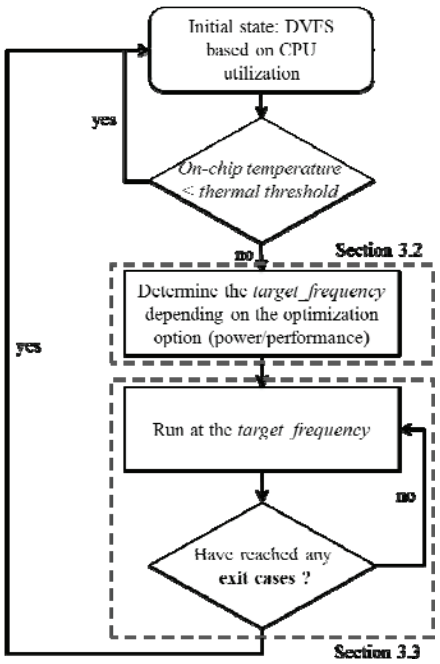

Fig. 1. An overview of the proposed temperature-aware DVFS scheme to co-optimize the power and performance.

3 Temperature-aware DVFS Scheme

In Section 3.1, we briefly introduce the overview of our proposed DVFS scheme. Then we explain how to determine the CPU frequency that co-optimizes the power and performance, in Section 3.2. Finally, we describe the frequency scaling method and the exit cases in Section 3.3.

3.1 Scheme Overview

Fig. 1 depicts our proposed DVFS scheme. At the initial state, our scheme monitors the on-chip temperature and CPU utilization. While the on-chip temperature remains under the predefined thermal threshold, our scheme determines the CPU frequency based on the CPU utilization. Same as the conventional DVFS scheme, our scheme scales the frequency up to the maximum frequency when the utilization exceeds utilization threshold (80% is used as default). On the other hand, when there is any lower frequency that keeps the utilization under the utilization threshold, it scales down to the lower frequency.

When the on-chip temperature increases up to the thermal threshold, our scheme applies the novel DVFS scheme to co-optimize the power and performance. Our scheme determines an optimal frequency depending on the optimizing option. We refer to this optimal frequency as *target_frequency*. The *target_frequency* is determined as a relatively higher value when the performance optimizing option is used, compared to when the power optimizing option is used. Once the *target_frequency* is determined, our scheme scales the CPU frequency to the *target_frequency* until it encounters any of the exit cases.

When our scheme is applied, only a few hundred instructions needs to be additionally executed per 100 ms

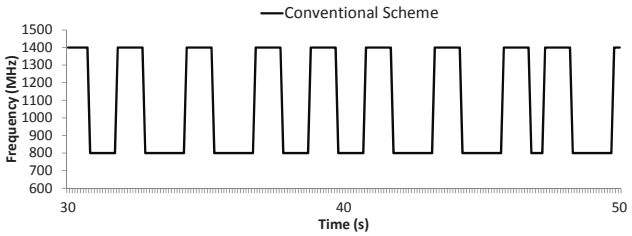

Fig. 2. The variation of CPU frequency when executing a program with the conventional DTM scheme.

(for either determining the *target_frequency* or scaling the frequency). Note that recent microprocessors can execute much more than a million instructions per 100 ms. Accordingly, our scheme brings up negligible performance overhead.

3.2 Frequency Determination Routine

3.2.1 Power Optimizing Option

When the power optimizing option is used, our scheme minimizes the power consumption, without performance degradation compared to the conventional DTM scheme. Fig. 2 shows an example of the frequency variation with the conventional DTM scheme, when the on-chip temperature increases up to the thermal threshold during the execution of a program that heavily utilizes the CPU. Since the CPU is heavily utilized, the conventional DVFS tries to scale the frequency to the maximum frequency (1400 MHz). However, due to the frequent DTM operation, the frequency is repeatedly scaled down to 800 MHz. Thus, the frequency is repeatedly changed between 800 MHz and 1400 MHz, as shown in Fig. 2. However, as we have briefly mentioned in Section 1, the unstabilized frequency, which implies unstabilized voltage as well, causes power inefficiency [3][8]. We formulate why frequency and voltage must be stabilized to achieve optimal power efficiency as follows.

- $Power \propto Frequency \times Voltage^2$ *(1)*
  Power consumption is proportional to voltage squared and frequency.
- $Voltage \propto Frequency$ *(2)*
  Voltage is proportional to frequency.
- $Power \propto Frequency^3$ *(3)*
  By Equation (1) and (2), power consumption is proportional to cubic frequency.
- $Average [F_1^3, F_2^3, F_3^3, \ldots, F_n^3] \geq Average [F_1, F_2, F_3, \ldots, F_n]^3$ *(4)*
  ($F_1, F_2, F_3, \ldots, F_n$ are the frequencies used at different time slices)
  Based on the Jensen's inequality [20], average of frequency cubes is larger than cube of average frequency, when the variance of the frequencies is not zero. Furthermore, the difference becomes larger as the variance increases. When there is no variation in the frequencies, both side of the equation has the same value.
- $Average [Power [F_1], Power [F_2], Power [F_3], \ldots, Power [F_n]] \geq Power [Average [F_1, F_2, F_3, \ldots, F_n]]$ *(5)*
  By Equation (3) and (4), average of power consumptions in different frequencies is larger than power consumption in average frequency. When the variance of the frequencies is zero, both side of the equation has the same value.
- $Performance \propto Frequency$ *(6)*
  Performance is proportional to frequency.

$\therefore$ From Equation (5) and (6), we confirm that running at the average frequency consumes less power compared to running at different frequencies, while achieving same performance.

Therefore, we set the *target_frequency* as the average frequency of the conventional DTM scheme, to save power. The detailed procedure is described in the left side of the dotted box in Fig. 3. When the temperature increases up to the thermal threshold, our scheme keeps track of the CPU frequency in the frequency-history window; 10 Hz and 6 seconds is used for the sampling rate and window size, respectively in our experiment[2]. Then our scheme waits until the frequency-history window is filled. In the meantime, the conventional DTM scheme is used. After the frequency-history window is filled, *target_frequency* is set as the average of the frequency-history window. The *target_frequency* set by this procedure is used for the power optimizing option.

[2] 10 Hz is the sampling rate of both the conventional DVFS scheme and the DTM scheme. Therefore, we can fully record all the varying frequencies by using 10 Hz sampling rate. 6 seconds is used in consideration of the scheme overhead and the thermal response. In some cases, 6 seconds may be short to fully reflect the thermal characteristics. In such cases, the *target_frequency* can be reconfigured later by the exit cases, which will be explained in Section 3.3.

3.2.2 Performance Optimizing Option

When the performance optimizing option is activated, there are some additional steps to configure the *target_frequency*. The right side of the dotted box in Fig. 3 shows these steps. Our scheme starts by running at the *target_frequency*, which is obtained from the power optimizing option [3]. While running at the *target_frequency*, our scheme monitors the on-chip temperature during a predefined sampling period to check the peak temperature. In case the peak temperature is less than (thermal threshold – temperature margin), our scheme increases the *target_frequency* by 10 MHz; sampling period and temperature margin is set as 6 seconds and 2 degrees for our evaluation, by considering on-chip thermal response to the frequency/voltage scaling. (We conducted thermal response test to confirm that the on-chip temperature does not increase by more than 2 degrees within the sampling period, when increasing the frequency by 10 MHz) On the other hand, when the temperature is same or larger than (thermal threshold – temperature margin), our scheme ends the frequency determination routine.

3.3 Actual Frequency Scaling and Exit Cases

After our scheme determines the *target_frequency*, it sets the CPU frequency as the *target_frequency* until it encounters any exit case. The dotted box in Fig. 4 describes the execution of our algorithm after determining the *target_frequency*. Though it would be best to scale the CPU frequency to exact *target_frequency*, it may not be possible since mobile devices only support discrete frequency levels (in the device used for our evaluation, frequency can be scaled as a granularity of 100 MHz). Therefore, our scheme utilizes the actual frequency scaling routine, described in the upper-part of the dotted box, in Fig. 4. At first, the frequency-history window is updated every 100 ms, which is same as in case of the conventional DVFS scheme. Then, the frequency scaling routine compares the average of recently selected frequencies in the frequency-history window to the *target_frequency*. When the average frequency is lower than the *target_frequency*, the routine scales the frequency to the lowest selectable frequency (provided by the hardware) that is higher than the *target_frequency*. Otherwise, frequency selection routine scales the frequency to the highest selectable frequency that is same or lower than the *target_frequency*. Consequently, our frequency selection routine alternately scales to two nearest frequencies provided by the hardware, in case the *target_frequency* itself is not provided by the hardware; for instance, 1100 MHz and 1200 MHz when the *target_frequency* is 1150 MHz. On the other hand, our scheme scales to the exact *target_frequency* when the target_frequency is provided by the hardware. In our paper, we refer to running with the frequency scaling routine as running at the *target_frequency*.

While the processor runs at the *target_frequency*, there are two cases where the processor should not run at the *target_frequency*, any more. We define the two exit cases as follows.

[3] Due to the hardware restriction, it is not easy to scale to the exact frequency. Actual frequency scaling will be described in Section 3.3.

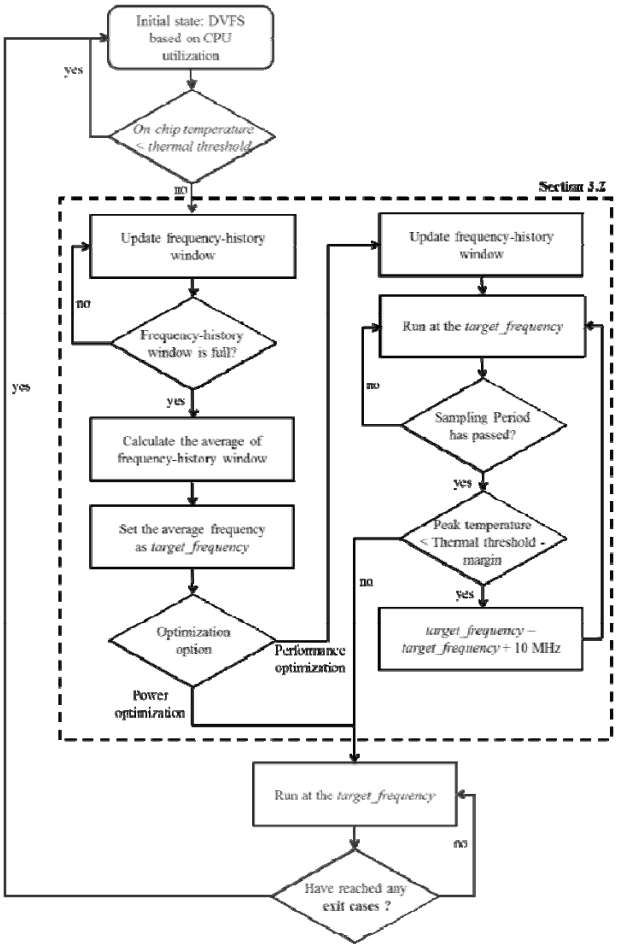

Fig. 3. The procedure to determine the *target_frequency*, depending on the optimizing option.

- Performance demand decreases: When the frequency which is lower than the *target_frequency* keeps CPU utilization under the utilization threshold, it is determined that the performance demand has decreased. In this case, the frequency should be scaled down to save power.
- On-chip temperature increases [4] up to the thermal threshold: In this case, the frequency should be scaled down for reliability.

[4] When the *target_frequency* properly reflects the thermal characteristics of the workload, the on-chip temperature should be stabilized at some point below the thermal threshold. However, the on-chip temperature may change during the execution due to one of the following reasons: (1) program behavior varies during the execution, and (2) thermal environment such as ambient temperature varies during the execution.

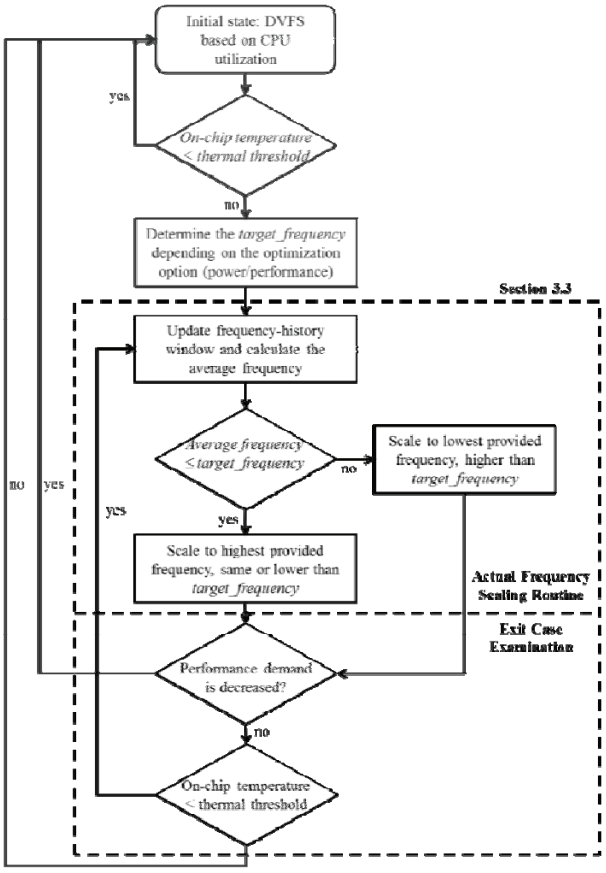

Fig. 4. The frequency scaling method and the exit cases at the *target_frequency*.

When our scheme encounters an exit case, it exits from the frequency scaling routine, consequently going back to the initial state. After returning to the initial state, frequency is reconfigured by the algorithm introduced in the Sections 3.1 and 3.2. In case that the performance demand has decreased, the frequency scales down to the lowest frequency that keeps the CPU utilization under the utilization threshold. On the other hand, when the temperature has increased to the thermal threshold, our scheme re-enters the frequency determination routine (which is described in Section 3.2) to set new *target_frequency*.

Table 1. EEMBC Benchmark Programs

Type	Application	Description
office	bezier01fixed	Bezier curve algorithm with integers
	bezier01float	Bezier curve algorithm with floating points
	dither01	Flyoid-Steinberg error diffusion dithering algorithm
	rotate01	Bitmap rotation algorithm
	text01	Text parsing algorithm
automative	matrix01	Matrix calculations
consumer	cjpeg	Standard compression algorithm for JPEG
	djpeg	Sandard decompression algorithm for JPEG
	rgbcmy01	Grey-scale filter algorithm
	rgbhpg01	RGB-to-CMYK conversion algorithm
	rgbyiq01	RGB-to-YIQ conversion algorithm
network	ospf	OSPF algorithm
	routelookup	Route lookup algorithm

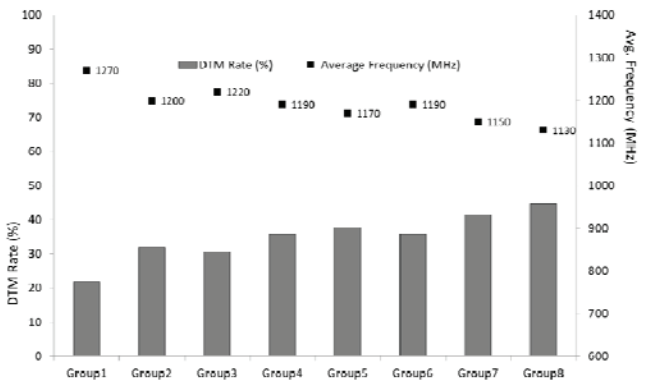

Fig. 5. DTM operation rate and average frequency of the **conventional DTM**.

4 Evaluation

4.1 Evaluation Methodology

4.1.1 Experimental Environments

For our evaluation, we use one of the high-end mobile devices, Odroid-Q [21]. Odroid-Q adopts a high-performance quad-core processor, Exynos 4412 [19], which provides the maximum frequency of 1.4 GHz and discrete frequency steps in granularity of 100 MHz. The device operates with Android ICS (Ice Cream Sandwich) and Linux 3.0.15 kernel. The system utilizes ondemand scaling governor [15] as the base power management policy (DVFS). When the on-chip temperature increases up to the thermal threshold, the conventional DTM scheme overrides power management policy (ondemand), scaling down the frequency to 800 MHz. When the temperature decreases under the thermal threshold, ondemand is applied again.

We evaluate two different versions of our scheme, (1) power optimizing option and (2) performance optimizing option, in the perspective of temperature, power and performance. In order to evaluate our scheme in the perspective of power, we use PowerMonitor [22] that keeps the log of system-wide power consumed by the device. At the same time, performance is evaluated on a real system. Additionally, we gain the thermal information from the on-chip thermal sensor, via the kernel message. During the evaluation, the ambient temperature was fixed as the normal temperature, at 19 degrees – in this paper we use Celsius temperature for all the thermal explanation.

4.1.2 Benchmark

For the evaluation of our scheme, we use 13 embedded benchmark programs from the EEMBC [18]. The programs are classified into four different types. The detailed description of each benchmark program is shown in Table 1. As each benchmark program is designed to run on a single core, we evaluate eight groups of benchmark programs, each composed of four different benchmark programs. Note that our target device has a quad-core processor. The combination of each group is determined arbitrarily to show the evaluation of mixed usage of benchmark programs. The detailed information of each group is shown in Table 2. With the default configuration of each benchmark programs, the execution time varies greatly, which makes it hard to evaluate the performance. To resolve this problem, we first execute each benchmark program on a single core, at the maximum frequency to measure the execution time. Note that DTM is not invoked in our evaluation environment, when utilizing just one core. Then, we adjust the number of iterations for each benchmark program so that all the execution times for benchmark programs are set as 10 minutes at the maximum frequency, when there is no DTM operation.

4.2 Experimental Results

4.2.1 Performance of the Conventional DTM

In this subsection, we describe the DTM rate and execution time of the benchmark groups with the conventional DTM. Fig. 5 shows the DTM operation rate during the execution of each group. DTM operation accounts for 21.6% (Group1) to 44.8% (Group8) of the total execution time. In general, the groups with more network benchmark programs (right side of the figure) have relatively higher DTM rate. On the other hand, groups only composed of the office and consumer benchmark programs have lower DTM rate. With the varying DTM rate, the average CPU frequency for each group also varies from 1270 MHz to 1130 MHz. The average frequency deservedly becomes lower as the DTM rate increases.

The increased DTM rate leads to the increased execution time of each group. Fig. 6 shows the execution time of the groups - note that we have configured the number of iterations so that each benchmark program can be completely executed in 10 minutes, when DTM operation is not invoked. Since the execution time of the 4 benchmark programs in each group may differ, we investigated two different execution times as follows: (1) the average execution time of all the benchmark programs in the group and (2) the longest execution time among the benchmark programs in the group. However, difference between the average and longest execution time is at most 10 seconds for all the groups. Therefore, we refer to the longest execution time as the execution time of the group for the rest of this paper. As the execution time is inversely proportional to the average frequency, the groups that invoke more DTM operations have longer execution time. In case of Group8 where DTM rate is as high as 44.8%, the execution time is also increased by 156 seconds compared to the execution without DTM operation (600 seconds). In case of Group1, the execution time is increased by only 78 seconds due to the lowest DTM rate (21.6%).

4.2.2 Power Optimizing Option

In this subsection, we evaluate our proposed scheme with the power optimizing option, explained in Section 3.2.1. Fig. 7 shows the execution time and system-wide energy consumption of our scheme, each normalized to that of the conventional DTM. As expected, the execution time is nearly the same as in the conventional DTM, since the power optimizing option sets the *target_frequency* as the average frequency of the conventional DTM. While achieving the similar performance, energy consumption is reduced by 12.7%, on average, and 16% in the best case (Group8). In general, the energy saving is larger for the groups with higher DTM rate. The results can be explained by the equations in Section 3.2.1 which describes that larger variance of frequencies worsens the power efficiency more significantly – note that the variance of frequencies becomes larger when the DTM rate becomes larger in the conventional DTM since the benchmark programs are executed at two different frequencies (normal frequency and much lower frequency caused by DTM invocation) for the most of the time. Therefore, our scheme saves more energy by stabilizing the frequency, for the groups with higher DTM rate.

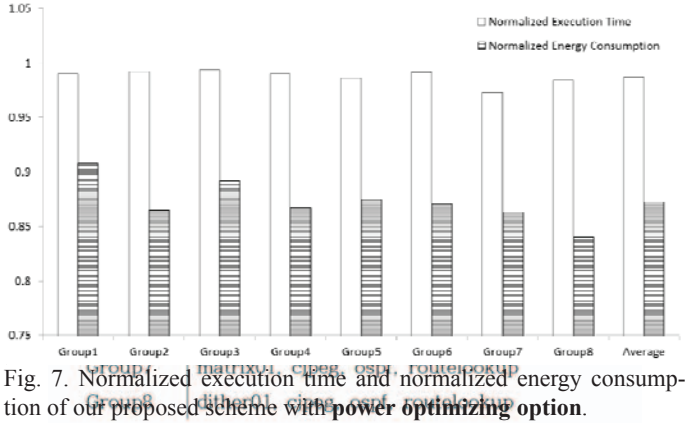

Fig. 7. Normalized execution time and normalized energy consumption of our proposed scheme with **power optimizing option**.

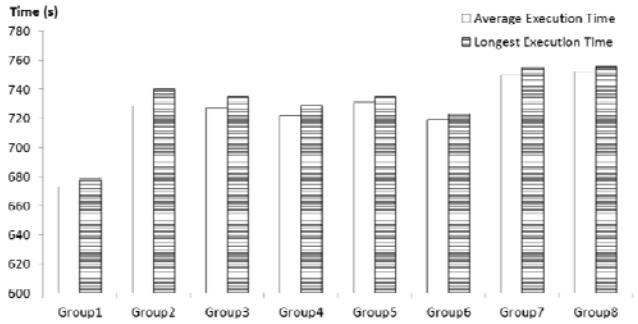

Fig. 6. Execution time of the **conventional DTM**.

In Fig. 7, the energy saving is relatively higher for Group2 when considering its lower DTM rate. This can be explained by the average frequency of the Group2 (1200 MHz); different from the average frequencies of the other groups, 1200 MHz is directly selectable in our target device. Thus, our scheme runs Group2 at the exact *target_frequency* without alternately scaling between two nearest frequencies – note, though our frequency scaling routine provides the sub-optimal power efficiency for other groups, it is optimal to scale to the exact *target_frequency*. Therefore, power saving for Group2 is relatively higher compared to groups with similar DTM rate.

When our scheme is applied with the power optimizing option, the on-chip temperature is also expected to decrease due to the energy reduction which is similar to average power reduction (note, there is no significant difference in the execution time). Fig. 8 shows the average temperature of each group when the conventional DTM is used and when our scheme is used with the power optimizing option. The results show that our scheme with power optimizing option reduces the on-chip temperature by 8.5 degrees, on average, compared to the conventional DTM. Furthermore, there is 9.6 degrees of margin between the average on-chip temperature of our scheme and the thermal threshold (86 degree). Not to mention the thermal reliability, the reduced temperature also leads to two additional benefits. Firstly, we may utilize the temperature margin to enhance the performance (which is our performance optimizing option explained in Section 4.2.3). Secondly, the reduced temperature leads to leakage power reduction, since leakage is dependent on temperature.

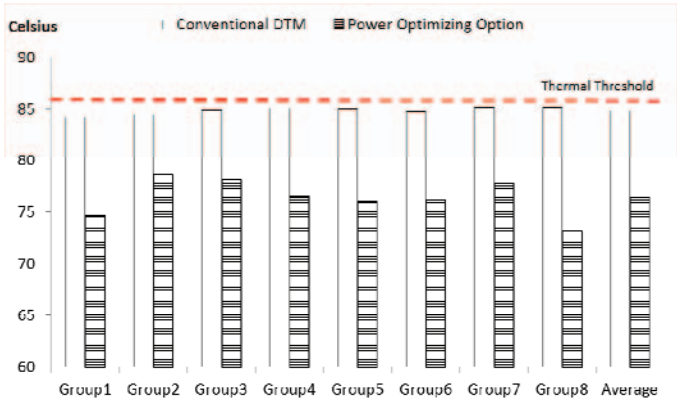

Fig. 8. Average on-chip temperature of the conventional DTM and our proposed scheme with **power optimizing option**.

4.2.3 Performance Optimizing Option

In this subsection, we evaluate our proposed scheme with the performance optimizing option. Fig. 9 compares the average frequency of our scheme and the conventional DTM. When our scheme is applied with the performance optimizing option, the average frequency is increased by 30 ~ 130 MHz, depending on the group. The average frequency varies due to the unique thermal characteristics of each group. Nevertheless, the frequency is increased for all the groups, resulting in performance enhancement.

Fig. 10 shows the execution time and system-wide energy consumption of our scheme with the performance optimizing option, each normalized to that of the conventional DTM. As the execution time is inversely proportional to the average frequency, our scheme reduces the execution time by average of 6.3% and maximum of 10% (Group8).

In addition to performance enhancement, energy consumption is also reduced compared to the conventional DTM. The average energy consumption is reduced by 6.7% owing to three different reasons: (1) stabilized frequency reduces power consumption compared to the unstabilized frequency, (2) lower on-chip temperature reduces the leakage power, and (3) reduced execution time leads to lower energy consumption since the energy can be calculated as average power multiplied by execution time. Group2 and Group3 mostly benefit from the first and second reason. Though the execution time is not decreased significantly, there is noticeable energy reduction. On the other hand, the other groups mostly save energy owing to the reduced execution time. Group6 may be the representative example; the average power consumption of our scheme with the performance optimizing option and the conventional DTM is 5602 mW and 5480 mW, respectively. Nevertheless, our scheme with the performance optimizing option saves energy by 3%, due to the 7% reduction in the execution time.

We also investigate the on-chip temperature to confirm that our proposed scheme with the performance optimizing option does not incur thermal problems. Fig. 11 shows the thermal results of our proposed scheme with performance optimizing option. The figure indicates that

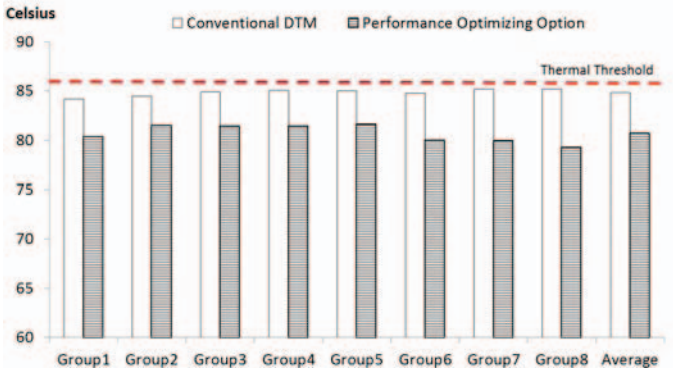

Fig. 11. Average on-chip temperature of conventional DTM scheme and our proposed scheme with **performance optimizing option**.

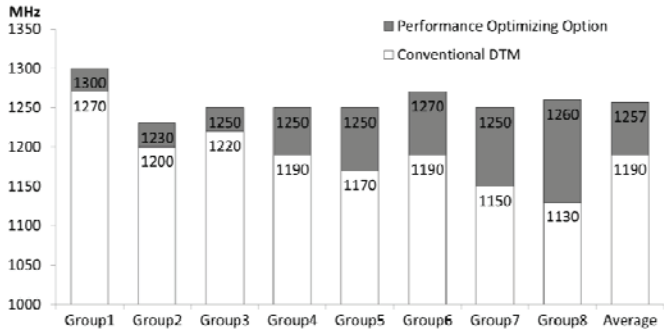

Fig. 9. Average frequency of our scheme with **performance optimizing option**.

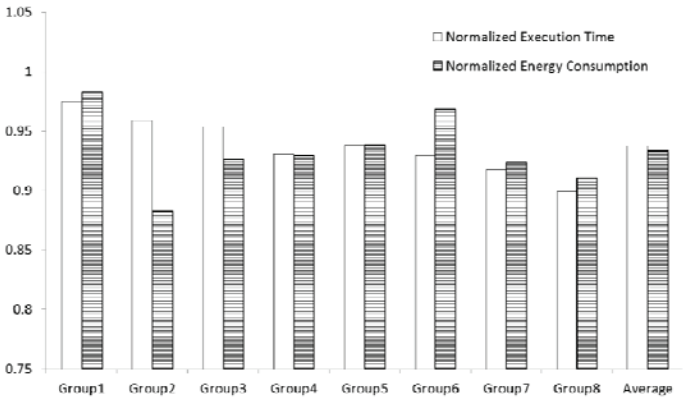

Fig. 10. Normalized execution time and normalized energy consumption of our proposed scheme with **performance optimizing option**.

average temperature of our proposed scheme is still 4 degrees lower than that of the conventional DTM. This result is mostly driven by the frequency determination algorithm itself; when determining the *target_frequency* for the performance optimizing, we stop increasing the frequency when the on-chip temperature is 2 degrees away from (lower than) the thermal threshold. Consequently, the average on-chip temperature is relatively lower compared to the conventional DTM, where DTM operations are invoked after the temperature increases up to the thermal threshold.

In summary, our scheme with the performance optimizing option improves the performance (6.3%) and energy (6.7%) at the same time compared to the conventional DTM. When compared to the power optimizing option, our scheme with the performance optimizing option enhances the performance by 6%, on average, though it consumes 5% more energy, on average.

5 Conclusion

The conventional DTM schemes used in off-the-shelf mobile devices monitor the temperature from the on-chip sensors to maintain thermal reliability. When the temperature is increased up to the thermal threshold, the schemes invoke DTM operation, which scales down the CPU frequency and voltage. However, the use of the conventional DTM schemes, results in not only inevitable performance degradation, but also power inefficiency. Based on the insight that the power efficiency can be improved at a stabilized frequency, we propose a temperature-aware DVFS scheme to co-optimize the performance and power.

Our scheme provides two different options: (1) the power optimizing option that saves energy, and (2) performance optimizing option that enhances the performance. We implement our scheme on an off-the-shelf mobile device to evaluate the scheme based on real measurement. While we evaluate the system power instead of the processor power, the energy saving is as much as 12.7% on average without any performance loss, by using our scheme with the power optimizing option. By using our proposed scheme with the performance optimizing option, performance is enhanced by 6.3% while the energy saving is still as much as 6.7% on average. In addition, our scheme also lowers the on-chip temperature compared to the conventional DTM, regardless of the optimizing option. Therefore, we believe that our scheme can be a viable solution for the mobile devices such as smartphones, where performance, power, and the temperature need to be considered together.

References

[1] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," *The 7th International Symposium on High-Performance Computer Architecture*. 2001. HPCA. pp.171-182. DOI=10.1109/HPCA.2001.903261

[2] A.P. Chandrakasan and R.W. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498-523, April 1995. DOI=10.1109/5.371964

[3] J. Chang and M. Pedram, "Energy Minimization Using Multiple Supply Voltages," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 5, no. 4, pp. 436-443, December 1997. DOI=10.1109/92.645070

[4] X.Y. Chen, K.C. Toh and J.C. Chai, "Direct Liquid Cooling of a Stacked Multichip Module," *Proc. of the 4th Electronics Packaging Technology Conference.* 2002. pp. 380-384. DOI=10.1109/EPTC.2002.1185702

[5] A.R. Cho, H.B. Jang, J.H. Lee, and S.W. Chung, "Reducing Leakage Power by Controlling Cooling Fan Speed," *COOLChips XIII,* (2010) (poster)

[6] H. Hanson, S. W. Keckler, S. Ghiasi, K. Rajamani, F. Rawson, and J. Rubio, "Thermal Response to DVFS: Analysis with an Intel Pentium M," *Proc. International Symposium on Low Power Electronics and Design.* (ISLPED '07), ACM. pp. 219-224. DOI=10.1145/1283780.1283827

[7] V. Hanumaiah and S. Vrudhula, "Temperature-Aware DVFS for Hard Real-Time Applications on Multicore Processors," *IEEE Transactions on Computers*. 2012. vol. 61, no. 10, pp. 1484-1494. DOI=10.1109/TC.2011.156

[8] T. Ishihara, H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *International Symposium on Low Power Electronics and Design*, pp.197-202, August 1998.

[9] H.B. Jang, I. Yoon, C.H. Kim, S. Shin, S.W. Chung, "The Impact of Liquid Cooling on 3D Multi-core Processors," *IEEE International Conference on Computer Design*. 2009. pp.472-478. DOI=10.1109/ICCD.2009.5413115

[10] J. Kong, S.W. Chung, and K. Skadron, "Recent Thermal Management Techniques for Microprocessors," *ACM Computing Survey*, June 2012. vol. 44, no. 3, Article 13, pp. 1-42. DOI= 10.1145/2187671.2187675

[11] J.S. Lee, K. Skadron and S.W. Chung, "Predictive Temperature-Aware DVFS," *IEEE Transactions on Computers*. 2010. vol. 59, no. 1, pp. 127-133. DOI=10.1109/TC.2009.136

[12] G. Liu, M. Fan, G. Quan, M. Qiu, "On-Line Predictive Thermal Management under Peak Temperature Constraints for Practical Multi-core Platforms," *Journal of*

Low Power Electronics. 2012, vol. 8, no. 5, pp. 565-578. DOI= 10.1166/jolpe.2012.1216.

[13] Y. Liu, H. Yang, R.P. Dick, H. Wang, and L. Shang, "Thermal vs Energy Optimization for DVFS-Enabled Processors in Embedded Systems,". *8th International Symposium on Quality Electronic Design*. 2007. pp.204-209. DOI=10.1109/ISQED.2007.158

[14] T. Nguyen, M. Mochizuki, K. Mashiko, Y. Saito and I. Sauciuc, "Use of Heat Pipe/Heat Sink for Thermal Management of High Performance CPUs," *Semiconductor Thermal Measurement and Management Symposium. Sixteenth Annual IEEE.* 2000, pp.76-79. DOI=10.1109/STHERM.2000.837064

[15] V. Pallipadi and A. Straikovsky, "The Ondemand Governor – Past, Present, and Future", *Proc. Linux Symposium.* July 2006, vol. 2, pp. 215-229.

[16] D. Shin, S.W. Chung, E.-Y. Chung and N. Chang "Energy-Optimal Dynamic Thermal Management: Computation and Cooling Power Co-Optimization," *IEEE Transactions on Industrial Informatics*. 2010. vol. 6, no. 3, pp. 340-351. DOI=10.1109/TII.2010.2052059

[17] K. Skadron, "Hybrid Architectural Dynamic Thermal Management," *Proc. Conference on Design, Automation and Test in Europe*. 2004, vol. 1, pp. 10-15.

[18] EEMBC, The Embedded Microprocessor Benchmark, available at: http://eembc.org

[19] Exynos 4412 by Samsung Electronics, available at: http://www.samsung.com/global/business/semiconductor/minisite/Exynos/index.html

[20] Jensen's Inequality, available at: http://en.m.wikipedia.org/wiki/Jensen's_inequality#section_2

[21] Odroid-Q by Hardkernel, available at: http://www.hardkernel.com/renewal_2011/products/prdt_info.php?g_code=G133888637376

[22] PowerMonitor by Moonson Solutions, available at: http://www.msoon.com/LabEquipment/PowerMonitor/