A case for bad big.LITTLE switching: How to scale power-performance in SI-HMP

Seehwan Yoo
Dankook University,
Republic of Korea
seehwan.yoo@mobile-os.dankook.ac.kr

YoonSeok Shim
Dankook University,
Republic of Korea
sys11@mobile-os.dankook.ac.kr

Seunghac Lee
Dankook University,
Republic of Korea
seunghack@dankook.ac.kr

Sang-Ah Lee
Dankook University,
Republic of Korea
sanga13@mobile-os.dankook.ac.kr

Joongheon Kim
Platform Engineering Group
Intel Corporation, California,
USA
joongheon.kim@intel.com

ABSTRACT

Recently, single-ISA heterogemeous multi-core processors (SI-HMP) draw attention, pursuing optimal power-performance scaling. Leveraging differently optimized heterogeneous cores, SI-HMP can dynamically tune performance with minimal additional power consumption, or it can find maximum performance core combination with respect to a given power budget. However, the little-to-big, or big-to-little core switching has hidden costs. To properly scale up/down the power-performance, we should carefully analyze the actual performance gain, considering the multi-core processing model and inter-cluster communication. This paper reveals that there are some good and bad cases for core switching, and presents a possible way to achieve good power-performance scaling through big-little switching.

Categories and Subject Descriptors

C.1.2 [**Computer Systems Organization**]: Parallel architectures—*multicore architectures*; C.1.4 [**Computer Systems Organization**]: Other architectures—*Heterogeneous systems*

Keywords

Single-ISA heterogeneous multi-core architecture, power- performance

1. INTRODUCTION

High-performance and energy-efficiency are two topmost demands for mobile processors. A user wants to play 3D graphic games smoothly, and wants to send / receive SMS messages for a long time without recharging the battery. Unfortunately, the two demands are difficult to be met at the same time because processor's power and performance are in trade-off relationship. Thus, scaling power and performance has been a hot topic in designing new microprocessors for last decades. As an evolutionary alternative to traditional approaches such as DVFS (Dynamic Voltage and Frequency Scaling), and dynamic core off-lining, single-ISA (Instruction Set Architecture) heterogeneous multi-core processor (Single ISA-HMP, SI-HMP) architecture draws attention.

Hotpower '15 October 04-07, 2015, Monterey, CA, USA
© 2015 ACM. ISBN 978-1-4503-3946-9/15/10...$15.00
DOI: http://dx.doi.org/10.1145/2818613.2818745

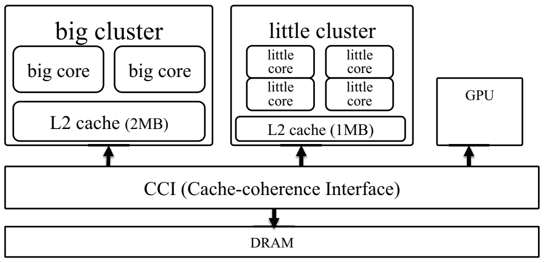

Figure 1: Big.LITTLE design with 2 big + 4 little cores

SI-HMP can scale up and down the power consumption and throughput, leveraging heterogeneous cores that are finely tuned to different performance goals such as low-power or high throughput. A SI-HMP includes multiple cores that share the same ISA, but the cores in the processor would have different microarchitecture, presenting different power and performance. For example, ARM introduced big.LITTLE design in which a microprocessor includes two different types of cores: big cores, that presents high throughput, and little cores, that presents low-power consumption.

Figure 1 shows a reference big.LITTLE processor design[2]. In the processor, there are two clusters: a big cluster and a little cluster. In the big cluster, there are two big cores (CortexA-57), and 2MB of shared L2 cache. In the little cluster, there are four little cores (CortexA-53), and 1MB of shared L2 cache. Figure 1 also shows the CCI (Cache-Coherent Interconnect)[1]. CCI manages cache coherency between separated L2 cache in two clusters. According to the protocol, each cluster's L2 cache can share data, and maintain updated value within each cluster's own cache.

A key mechanism in power-performance scaling is inter-cluster core switching (or big-little switching)[1]. Big-little switching trades power with performance by exchanging the core configurations. For example, to increase throughput, sacrificing the power efficiency, one would exchange an actively running little core with a big core. This paper focuses on the big-little core switching effect on SI-HMP power-performance scaling. In general, little-to-big switching is believed to be a scaling up operation.

However, big-little switching does not always present a good power-performance scaling. To properly scale with parallel or multi-core workloads, we should additionally consider parallel processing model, and inter-cluster communication.

Big-little switching should be carefully performed, considering the bottleneck performance. When we exchange little core with big core, the remaining slow little cores would be performance bottleneck. This results in only marginal performance scaling by big-little switching.

In addition, big-little switching could also incur sequelae, unexpected a performance degradation after the switching. That happens particularly often when multi-threaded application is running. When a communicating thread shares data with another core in a different cluster, then the communication cost becomes significant in order to maintain per-cluster L2 cache coherency. Moreover, synchronization such as spin lock is critical in performance. The big-little switching would end up with additional power consumption without any performance enhancement.

A goal of this paper is to evaluate the big-little core switching as a means of performance scaling, revealing the actual configuration for possible scaling. Our results under application scenarios identify there is a bad inter-cluster core switching, or inverse performance scaling, could happen.

The rest of the paper is structured as follows. Section 2 presents related study on SIHMP, and power-performance scaling in mobile processors. Section 3 presents an optimistic view of big-little core switching with CPU-intensive benchmark. Section 4 shows some results with some parallel benchmarks, and spinlock, highlighting the bad cases of inter-cluster core switching. Section 5 concludes the paper with some possible future work.

2. RELATED WORK

Big.LITTLE design has several variant implementations according to the core combination, cache coherency support, and scheduling inside the OS. Hardware architecture is important in recent power-performance study because different architecture presents different power and performance characteristics. We use Juno platform from ARM [2] that incorporates 2 big cores and 4 little cores, as shown in Figure 1. Juno processor is featured with separated per-cluster L2 cache, and the CCI that manages coherency between the caches. In fact, CCI manages coherency between bus masters, such as core cluster, GPU, NIC. CCI is outside of the core cluster; thus, CCI registers are not directly operated on CPU. Accordingly, all CCI operation is regarded as bus transaction, happens beneath the L2 cache.

Hahnel and Hartig presented system-wide energy consumption in a mobile development platform[6]. In the study, the authors measured energy consumption by different hardware components for several workloads. The analysis is very helpful to understand the network, storage-using application could be affected by big.LITTLE scheduling. In addition, they present the cluster-switching impact to energy consumption. The result reveals that cluster-switching cost is substantial particularly when memory heavy applications is used. Their processor, unfortunately does not fully support cache-coherency between big and little cluster. Therefore, their model has inherent heavy cluster switching cost because all the cores in the cluster have to move with their architectural status, such as cache, TLB, etc. Our big.LITTLE platform fully supports cache coherency, and it supports per-task migration as well as per-core, and per-cluster switching. Thus, our study extends the previous migration study with CCI-related inter-cluster communication cost.

Heterogeneous computing architecture has been explored by several studies. Shen et al. exploit programmable DSP to offload computation-intensive module from the application processor [10]. The authors propose an Android framework so that android applications and services can take advantage of it. More radical approach, Felix et al. present an approach to use similar-ISA heterogeneous multi-core architecture[8]. In the study, the authors present dual core ARM cpu (dual Cortex-A9) can loosely coupled with co-processor that supports ARM ISA (Cortex-M3). The co-processor has 1/5 performance numbers, but it presents 1/17 power consumption. To loosely couple the heterogeneous cores, the authors proposed distributed runtime environment, called Kage that maintain replicas in two different domains. As a research prototype, they present interesting approach, but there are several practical consideration such as programmability, general applicability to DSP offloading, application transparency in two distributed domains with responsiveness, scheduling /inter-process communication across the heterogeneous domains.

In recent mobile systems, energy consumption by the main processor takes about 20% of the entire system, which means that CPU-only energy-saving solutions could be suboptimal. Yet, the system performance and energy consumption are highly dependent upon the utilized hardware components. That is, when the workload is CPU-intensive, we still have room for additional improvements. Although the followings are CPU and GPU-specific studies, they present a possible improvements under some application scenarios. DVFS has been widely adopted technique in mobile devices. Xinxin et al. present an approach to use DVFS on GPU[9]. Taking advantage of DVFS, energy consumption by GPU can be considerably reduced. Along with the CPU power management, GPU DVFS can be effectively used. Yifan refined the power model of mobile CPU[11]. In the model, the authors incorporate multiple-levels of CPUIdle state. Aaron et al. casted a serious question on mobile multi-core[4]. The authors measure energy for different workload, with different processors. For some processors, idle power consumption is very small, and it is infeasible to scale down power-performance. Their recent paper addresses unified approach to utilize DVFS and core offline[5].

Recent Linux has cpufreq driver that controls DVFS level according to the given hardware platform[7]. It also has cpuidle driver that changes cpuidle states (C-states or P-states) according to the wake/sleep duration. Linux also has governor in power management framework. The governor

[1]In this paper, we will use core switching or big-little switching, instead of inter-cluster core migration to avoid confusion with task migration.

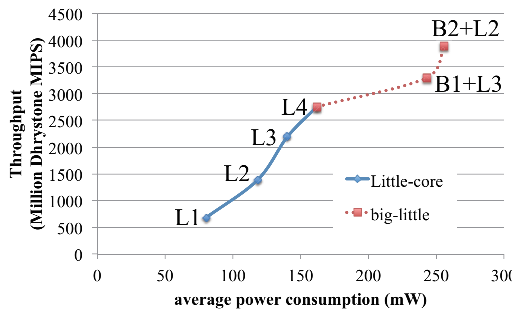

Figure 2: Big.LITTLE scaling with MPWhetstone

can specify the system-wide power management policy. It uses cpu-hotplug framework, which dynamically controls on-line and off-line cores in run-time. These cpufreq/cpuidle drivers and governor closely interplay with the scheduler. Recent big.LITTLE scheduler tracks the CPU intensity of a task, and migrates a task to a big core if it has high intensity.

3. SCALING WITH CPU INTENSIVE WORKLOAD

Big and little cores present different power-performance, and inter-cluster core switching tries to take the best advantage of it. To present the net benefit from core switching, we run Whetstone, that is a CPU-intensive benchmark, on each core varying the number of cores.

The hardware platform is ARM's reference big.LITTLE platform [2], and used kernel is the latest Linaro kernel. The core clock is set to maximum 850Mhz for little cores, 1.1Ghz for big cores, to comparatively present the big and little performances.

Figure 2 shows the result. In the graph, X-axis is average power consumption during execution. Y-axis is throughput (Million Whetstone Instructions Per Second). In the little cores graph (solid line), power and performance scales up as we put more cores. It scales very linearly because there is no interdependency between threads, and each of them does its own calculation. After four little cores allocation, we exchange little cores with big cores, assuming that a big-little core switching happens (dotted line).

In Figure 2, compared with 4 little cores case, big2+little2 cores present 40% improved throughput from 2756.60 MWIPS (Million Whetstone Instruction Per Second) to 3887 MWIPS, with 58% additional power consumption (from 161.96 mW to 255.9 mW). Big1+little3 cores combination present 20% improved throughput from little4 cores in MWIPS, with 49 % of additional power consumption.

In the graph, big1+little3 cores throughput increases almost linearly, but power consumption has been much more increased. A reason is that a big core consumes more power than a little core because it has larger cache, TLB, complex pipeline structure, etc. In addition, power consumed by per-cluster hardware resources (L2 cache, interrupt distributer, snooping control unit, etc.) is introduced by turning on the first core in the big cluster. When we use more big core (two big cores and two little cores), power consumption marginally increases, and the throughput increases sharply. It shows almost linear scaling when two big cores are used. The result supports that big.LITTLE core switching can possibly extend the positive scaling of power-performance.

Table 1: Facesim performance with big-little switching

	4L	1B+3L	2B+2L
exec.time (s)	911	895	861
avg. power (mW)	235.25	255.12	264.64

4. SCALING WITH PARALLEL TASKS

4.1 facesim

Beside the Whetstone, we run some PARSEC benchmark[3], that provides parallel workloads, assuming that those benchmark can best utilize multi-core hardware.

At first, we run facesim in PARSEC. Facesim simulates the facial motion for realistic emotion expressions. The simulation consists of several phases. At the beginning of benchmark, the benchmark process makes multiple threads, and the process divides a cpu job into small pieces, as many as the number of threads. Then, each thread performs the simulation, independently. At the end of each phase, the calculation results are gathered, and the next phase begins.

To compare the performance scaling according to big-little switching, we run facesim with different four cores-combinations: 4 little cores (4L), 1 big core + 3 little cores (1B+3L), 2 big cores + 2 little cores (2B+2L), assuming the big-little core switching. Then, we measure the execution time and average power consumption during the execution

In Table 1, 2B+2L presents a little 5.8% improvements in throughput, (reduced execution time from 911 s $\rightarrow$ 861 s) with additional 12.5% power consumption (235 mW $\rightarrow$ 264 mW), by exchanging 2 little cores with 2 big cores. In addition, 1B+3L presents only 1.8% of improved throughput with 8.4% additional power consumption. That is, it has negligible performance gain, not only in terms of performance but also in terms of power consumption.

A reason is that PARSEC facesim wrongly distributes parallel workload across big-little cores. Because a little core runs slower than a big core, if the same amount of job is given to two different (big and little) cores, a little core becomes the performance bottleneck, and a big core would be idle after the completion, which is observed from our execution traces. Facesim seems to distribute the same amount of work to all threads, regardless of the running core (big or little). In this case, a job is divided into 4 small pieces, and each thread runs one of 4 pieces. Thus, in all cases, little cores have high utilization ($> 90\%$), but big cores are seriously underutilized ($\sim$40%). Accordingly, the execution time is bound to the slowest little core. Thus, regardless of the core combination, the execution phase is determined by the little core's execution time for 1 piece of the job.

We additionally measured the power consumption and execution time with different core-combinations: 2 big cores (2B), 2 big + 2 little cores (2B+2L), and 2 big + 4 little (2B+4L) cores. If we divide the job with only two big cores, a job is divided into two pieces, and each big core is fully utilized, completing the given work at almost the same time. For 2B+4L case, a job is divided into 6 pieces, and each core performs 1/6 of the entire job.

The measured execution time and power consumption is given in Table 2. For convenience, we additionally put the normalized power consumption and normalized throughput along with big/little cores' utilization ($U._B$ and $U._L$), respectively. To compare normalized performance number, we additionally measure the execution time and power con-

Table 2: Facesim for proper scaling with parallel tasks

	exec. time(s)	power (mW)	norm. thruput	norm. power	U.$_B$ (%)	U.$_L$ (%)
L1	666	86.7	1	1		100
B2	159	292.3	4.2	3.4	95.88	
B2+L2	**173**	**277.7**	**3.8**	**3.2**	49.3	88.4
B2+L4	131	372.5	5.1	4.3	47.5	88.6

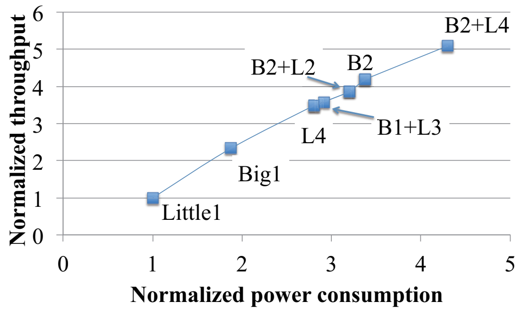

Figure 3: Big.LITTLE switching with facesim

sumption for one little core (L1).

Observe that B2+L2 has longer execution time than B2 even though B2+L2 uses additional little cores. That is because big cores in B2+L2 waits until the completion of slow little cores. Due to the idle time of B2, the actual power consumption of B2+L2 is smaller than B2. Remember that a big core consumes more power than a little core.

B2+L4 presents better scaling than B2+L2. In the case, each core performs so small job not enough to saturate the CPU utilization; thus, the result presents improved the performance. Yet, big cores' utilization is still low, seemingly a room for additional improvements.

Figure 3 summarizes the performance of facesim, scaling with various cores combinations. Despite subtle inverse scaling (B2 vs. B2+L2), the scaling can be possibly achieved. Note that little-to-big core switching does not be much helpful for scaling (i.e. L4$\rightarrow$ B1+L3 $\rightarrow$ B2+L2) because remaining slow little core becomes the bottleneck.

Dynamic task migration across cores will help to alleviate the inefficient scaling. By moving a task that is running on a slow little core to a fast big core, a task can boost the execution temporarily.

4.2 swaption

In addition to facesim, we run swaption in PARSEC benchmark. Swaption performs monte-carlo simulation for price calculation of swaption portfolio. The benchmark makes multiple threads, and the threads communicate each other, sharing large amount of data (several virtual memory pages). Unlikely the facesim, it does not consists of several phases. Instead, it statically distributes jobs to multiple threads. When a thread completes the given execution, the task retires, and the task does not execute any work. Thus, CPU utilization is very high at the beginning, and the utilization becomes smaller as threads complete execution.

Table 3 shows the execution time of the swaption for different cores combinations: 4 little cores (4L), 1 big core + 3 little cores (1B+3L), 2 big cores + 2 little cores (2B+2L). As a comparison, the result of 2 big core (2B) case is also added in the table.

In Table 3, 2B+2L presents little (about 2.5%) improvements in throughput, (reduced execution time from 532 s $\rightarrow$ 519 s) with additional 54.8% power consumption (262 mW $\rightarrow$ 406 mW), by exchanging 2 little cores with 2 big cores. In addition, 1B+3L presents 2.3% of improved throughput with 23% additional power consumption. That is, big-little core switching does not work well, and end up with additional power consumption for the same performance. Moreover, B2 presents smaller execution time than B2L2, and power consumption of B2 is less than B2L2. Namely, inverse scaling also occurs here.

To investigate the reason of this inefficient scaling, we compare the profile information from L4 and B2L2, which is a big-little core switching, exchanging 2 little cores with 2 big cores. Then, we measure branch mis-prediction, instruction barrier operations, bus transactions on cluster-shared memory.

B2L2 shows significant number of barrier operations, bus transactions, branch mispredictions. A reason is that multiple threads use exclusive memory access instructions such as LDREX/STREX, CLREX. Those instructions allow a thread to have exclusive memory access by locking the bus. When a core holds a lock, memory operation is performed only on the core, making L2 cache dirty. In B2L2 case, after the exclusive memory operation, bus access occurs in order to keep the coherency between L2 caches in different clusters. Therefore, large number of bus transactions are measured. Comparatively note that L4 does not require bus access because all the data are on the local L2 cache.

In addition to exclusive memory access instructions, B2L2 presents frequent barrier operations such as ISB, DMB, DSB. Barriers are software-based methods to preserve the order of memory access among multiple threads. Memory barrier operation is costly because it could stall the pipeline to flush the write buffer. Along with the shared bus access, synchronization instructions execution, high branch mis-prediction events are observed in B2L2, which negatively affects the performance of B2L2.

4.3 spinlock

Inter-cluster core communication overhead could be more serious than expected. To illustrate the communication overhead, we experiment spinlock in two different cores, one in a big core, and another in a little core. The two threads race to increase the shared variable up to two hundred million. The execution time is comparatively measured with two threads run on big cores and little cores, respectively.

In Table 4, the execution time with big + little cores are almost twice longer than 2 big cores case, and even longer than spinlock with two little cores. The shared variable is cached in per-core L1 and per-cluster L2 cache. When the spinlocking cores are in the same cluster, shared L2 cache can be effectively utilized between cores. However, in big + little cores spinlock case, shared data in L2 cache is continuously invalidated by another cluster's core. Although CCI keeps coherency, there is a definite overhead when there is intensive communication.

4.4 summary

Table 3: Swaption execution time and IPC

	4L	1B+3L	2B+2L	2B
exec. time (s)	**532**	**521**	**519**	496
Avg. power (mW)	**262**	**323**	**406**	321
CPU utilization (%)	99.9	99.9	99.8	99

Table 4: 200M spinlock time on different cores

exec. time (s)	2 little	big + little	2 big
	167	288	114

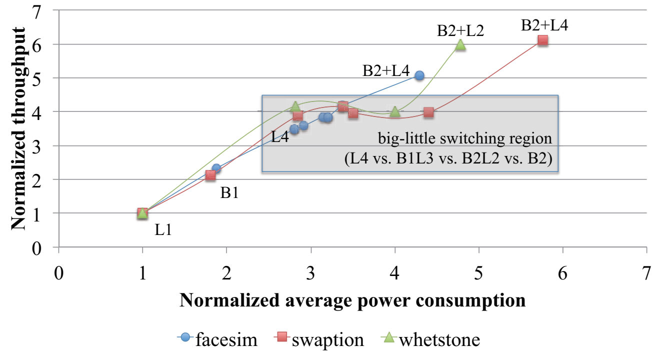

Figure 4: Power-performance scaling with swaption, facesim, whetstone

Figure 4 summarizes the power and performance of the whetstone, facesim, and swaption, scaling with various cores combinations. Overall, there is a smooth tendency of power-performance scaling with diverse cores combination. Small number of little cores present less power consumption with limited performance numbers. When all cores are used, performance increases along with the number of cores, with additional power consumption.

Observe that little-to-big core switching does not be much helpul for scaling (i.e. L4→ B1+L3 → B2+L2) because additional performance gain is limited by the remaining bottleneck performance and inter-cluster communication. Specifically for swaption case, the big-little switching shows almost same throughput with high power consumption, which presents a bad power-performance scaling. Additionally note that facesim and swaption results show that B2 present better throughput and less power consumption than B2L2.

5. CONCLUDING REMARKS

In this paper, we analyzed the power-performance scaling with respect to SI-HMP. Big.LITTLE has been proposed to scale power- performance by adopting single-ISA heterogeneous multicore. It has definite advantage because existing software can be transparently migrated one from another. However, the core switching could be costly. This paper evaluates big.LITTLE core switching with several benchmarks. General results are as follows: First, overall power-performance scaling can be achieved by diverse cores combination. Second, to scale with parallel applications, parallel processing model has to be considered. Despite the core switching, remaining little cores are easy to be a performance bottleneck. Third, some applications do not scale well, particularly when the threads make intensive inter-cluster communication. Our results show that core switching could support positive scaling or could end up with additional power consumption. As future work, we are investigating multi-thread applications and operating systems impact to evaluate feasibility of big-little switching on mobile platforms.

6. ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2015R1C1A1A02037330).

7. REFERENCES

[1] ARM. Amba axi and ace protocol specification. Accessed: 2015-07-26.

[2] ARM. Juno arm development platform. Accessed: 2015-07-26.

[3] Bienia, C., Kumar, S., Singh, J. P., and Li, K. The parsec benchmark suite: Characterization and architectural implications. Tech. Rep. TR-811-08, Princeton University, January 2008.

[4] Carroll, A., and Heiser, G. Mobile multicores: Use them or waste them. In *Proceedings of the Workshop on Power-Aware Computing and Systems* (New York, NY, USA, 2013), HotPower '13, ACM, pp. 12:1–12:5.

[5] Carroll, A., and Heiser, G. Unifying dvfs and offlining in mobile multicores. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th* (April 2014), pp. 287–296.

[6] Hähnel, M., and Härtig, H. Heterogeneity by the numbers: A study of the odroid xu+e big. little platform. In *Proceedings of the 6th USENIX Conference on Power-Aware Computing and Systems* (Berkeley, CA, USA, 2014), HotPower'14, USENIX Association, pp. 3–3.

[7] Holla, S. "acpi low-power idle states (lpi). In *Proceedings of the 6th USENIX Conference on Power-Aware Computing and Systems* (2015), Linux Plumbers conference. Accessed: 2015-07-31.

[8] Lin, F. X., Wang, Z., and Zhong, L. Supporting distributed execution of smartphone workloads on loosely coupled heterogeneous processors. In *Proceedings of the 2012 USENIX Conference on Power-Aware Computing and Systems* (Berkeley, CA, USA, 2012), HotPower'12, USENIX Association, pp. 2–2.

[9] Mei, X., Yung, L. S., Zhao, K., and Chu, X. A measurement study of gpu dvfs on energy conservation. In *Proceedings of the Workshop on Power-Aware Computing and Systems* (New York, NY, USA, 2013), HotPower '13, ACM, pp. 10:1–10:5.

[10] Shen, C., Chakraborty, S., Raghavan, K. R., Choi, H., and Srivastava, M. B. Exploiting processor heterogeneity for energy efficient context inference on mobile phones. In *Proceedings of the Workshop on Power-Aware Computing and Systems* (New York, NY, USA, 2013), HotPower '13, ACM, pp. 9:1–9:5.

[11] Zhang, Y., Wang, X., Liu, X., Liu, Y., Zhuang, L., and Zhao, F. Towards better cpu power management on multicore smartphones. In *Proceedings of the Workshop on Power-Aware Computing and Systems* (New York, NY, USA, 2013), HotPower '13, ACM, pp. 11:1–11:5.