Big or Little: A Study of Mobile Interactive Applications on an Asymmetric Multi-core Platform

Wonik Seo, Daegil Im*, Jeongim Choi, and Jaehyuk Huh
School of Computing, KAIST **Samsung Electronics*
{*wiseo, dgim, jichoi, jhhuh*}*@calab.kaist.ac.kr*

***Abstract*—This paper characterizes a commercial mobile platform on an asymmetric multi-core processor, investigating its available thread-level parallelism (TLP) and the impact of core asymmetry on applications. This paper explores three critical aspects of asymmetric mobile systems, asymmetric hardware platform, application behavior, and the impact of scheduling and power management. First, this paper presents the performance and energy characteristics of a commercial asymmetric multi-core architecture with two core types. The comparison between big and little cores shows the potential benefit of asymmetric multi-cores for improving energy efficiency. Second, the paper investigates the available thread-level parallelism and core utilization behaviors of mobile interactive applications. Using popular mobile applications for the Android system, this paper analyzes the distinct TLP and CPU usage patterns of interactive applications. Third, the paper explores the impact of power governor and CPU scheduler on the asymmetric system. Multiple cores with heterogeneous core types complicate scheduling and frequency scaling schemes, since the scheduler must migrate threads to different core types, in addition to traditional load balancing. This study shows that the current mobile applications are not fully utilizing the asymmetric multi-cores due to the lack of TLP and low computational requirement for big cores.**

***Keywords*-asymmetric multi-core, mobile application, scheduling, power management, workload analysis**

I. Introduction

Mobile platforms have emerged as dominant computing devices with the proliferation of smartphones and tablets. To accommodate the increasing demand for responsiveness and energy efficiency in such mobile platforms, mobile processors have evolved to multi-cores with more than two cores. Furthermore, commercial mobile multi-cores have adopted core asymmetry with two types of cores, big performance-optimized cores and little energy-optimized cores. Such asymmetric multi-cores with a single ISA have been proposed to improve the energy efficiency by selectively using a core with the best efficiency for a given task [1]–[3]. The current mobile asymmetric multi-cores are the first widely deployed commercial realization of core asymmetry.

With the complex asymmetric core capability available in the mobile processors, understanding the characteristics of mobile applications and systems has become critical to reduce energy consumption while providing interactivity of mobile applications. Furthermore, in energy-constrained mobile systems, dynamic voltage and frequency scaling (DVFS) has been extensively used. With core asymmetry combined with DVFS, each core type has its own performance and power variation with different frequency settings, and core asymmetry further complicates the scheduling and power management to optimize the system energy consumption and responsiveness for interactivity.

Therefore, to fully utilize the asymmetric multi-cores, understanding the characteristics of interactive mobile applications is crucial. With the first generation commercial mobile asymmetric system, this study characterizes the available parallelism and computational requirements of mobile applications for asymmetric multi-cores. This work investigates three important aspects of mobile asymmetric systems, the characteristics of asymmetric processor architecture, the thread-level parallelism and CPU usage patterns of interactive mobile applications, and the impact of scheduling and power management by the OS.

First, we evaluate the performance and power characteristics of a state-of-the-art mobile asymmetric multi-core architecture. Using mobile interactive applications along with CPU-intensive SPEC applications, we quantify the performance gain of big cores and energy reduction of small cores, comparing the two core types with various frequency settings. Second, we investigate the available parallelism and CPU usage patterns of mobile applications. Although the target asymmetric multi-core processor has four small cores and four big cores, only a fraction of the cores are active mostly. We quantify the parallelism by core type, and how often big cores are selected over small cores by the current scheduling scheme. Finally, we explore the effect of scheduler and power governor for energy and performance of mobile platforms. By adjusting critical parameters in the current utilization-based asymmetric scheduler and frequency governor, we explore the impact of scheduler and power governor on mobile systems.

Although the current asymmetric multi-cores open a new level of energy reduction while supporting high performance when needed, our observation shows multiple big cores are not fully utilized, but one or two big cores can improve the responsiveness of some applications significantly. Furthermore, mobile applications frequently under-utilize little cores at the lowest clock speed due to low CPU loads, and

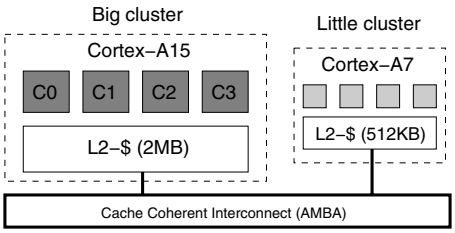

Figure 1. 8-core asymmetric multi-cores

Processor type	Value
Big core	4 x Cortex-A15(ARMv7 ISA), up to 1.9GHz Out-of-Order core with 15-24 stage pipeline 3-issue width, 3-fetch width L1 I/D-Cache: 32KB/2-way/64B L2 Cache: 2MB/16-way/64B
Little core	4x Cortex-A7(ARMv7 ISA), up to 1.3GHz In-order core with 8-10 stage pipeline 2-issue width, 2-fetch width L1 I-Cache: 32KB/2-way/32B L1 D-Cache: 32KB/4-way/64B L2 Cache: 512KB/8-way/64B

Table I
ARCHITECTURAL DETAILS OF BIG/LITTLE CORES

special hardware such as video decoding further reduces the CPU loads. Such an observation leads to a possible efficiency improvement with different core organizations and special hardware components in next-generation mobile asymmetric multi-cores.

The rest of this paper is organized as follows. Section 2 describes the methodology, and Section 3 evaluates the power and performance characteristics of mobile asymmetric multi-cores. Section 4 presents the prior work and background for asymmetric scheduler and power governor. Section 5 presents the TLP and utilization analysis of mobile interactive applications. Section 6 evaluates the impact of scheduling and power management with DVFS. Section 7 concludes the paper.

II. METHODOLOGY

To analyze the performance and power behavior of asymmetric multi-core platforms, we use Galaxy S5 as our target device, with the Exynos 5422 CPU which includes four ARM Cortex-A7 cores (little core), four ARM Cortex-A15 cores (big core), and a GPU. Figure 1 presents the overall architecture of the asymmetric processor, and Table I describes the architectural details of two core types. In the current implementation, any combination of big and little cores can be active, unlike the limitation of the previous big-little implementation, which allowed only either big or little cores, but not both types of cores, can be active at a time. However, one restriction is that one little core must be always active. Furthermore, as shown in Figure 1, the two core types have separate L2 caches. The big cores share a large 2MB L2 cache, but the little cores share a modest 512KB L2 cache. The two caches can be active simultaneously with coherence support. This difference in the L2 cache sizes leads to further performance difference between big and little cores larger than the prior studies suggested.

App Name	Description	Perf. Metric
PDF Reader	Open and read a pdf file	Latency
Video Editor	Edit a video file	Latency
Photo Editor	Edit a photo	Latency
BBench [4]	Run bbench on chrome browser	Latency
Virus Scanner	Scan applications and storages	Latency
Browser	Visit a site on chrome browser	Latency
Encoder	Encode a file	Latency
Angry Brid	Shooting game with physics engine	FPS
Eternity Warriors 2	3D action RPG game	FPS
FIFA 15	3D sport game	FPS
Video Player	Play a video file	FPS
Youtube	Search and play a video	FPS

Table II
MOBILE BENCHMARK APPLICATIONS

The target system runs Android 4.4.2. The default asymmetric scheduler is the HMP (Heterogeneous Multi-Processing) scheduler derived from the Linaro system. The power governor for DVFS is the `interactive` governor commonly used for mobile platforms. For power management, each core type must have the same frequency setting. The big core can have a frequency from 0.8GHz to 1.9GHz, and the little core can have from 0.5GHz to 1.3GHz. To measure the power consumption, the Monsoon power meter is used, and we measure the whole system power, including the rest of components beyond CPUs.

We have selected a set of 12 mobile applications for the Android platform for this paper. Table II shows our benchmark applications. Game applications, `video player`, and `youtube` generate a certain amount of CPU loads continuously, although users can change the execution path and pattern. In the rest of applications, CPU loads are initiated by the user interaction, such as touch inputs, with a strong burst of CPU load by user inputs. In Section 3, we also use two extra workloads to compare big and little cores. First, we use the SPECCPU2006 benchmark as CPU-oriented applications with high CPU utilization. We use the SPEC benchmark to focus on the architectural difference of the two core types. Second, we use a micro-benchmark which can adjust CPU utilization levels. We use the microbenchmark to analyze the power behavior of big and little cores with different CPU utilization levels.

To define the performance of mobile interactive applications, we use latency or frames-per-second (FPS), depending on applications. For a subset of applications, such as browsing, the latency can be defined as the time to complete a sequence of user actions. For those applications, we use the execution latency as the performance metric. However, for games, the performance of an application is defined only by how responsive the application is. In such cases, we use FPS as the performance metric. Table II shows the performance

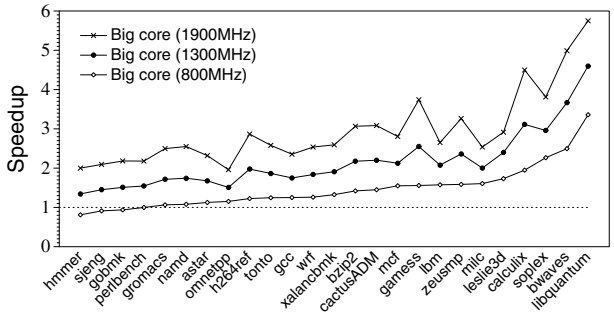

Figure 2. Speedup normalized to a little core with 1.3GHz

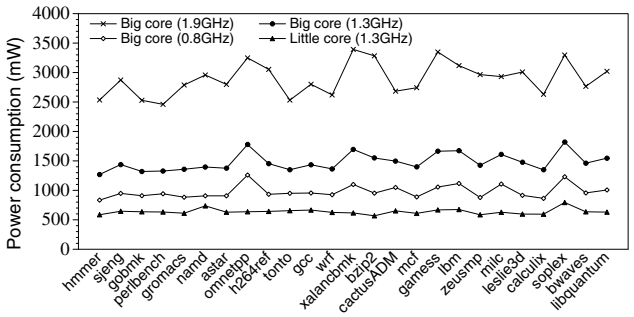

Figure 3. Power consumption (mW) for SPEC applications

metric for each application.

III. Architectural Characteristics

This section presents the hardware characteristics of the target mobile asymmetric multi-cores. We first compare the power and energy characteristics of big and little cores with the CPU-oriented SPECCPU applications and mobile interactive applications. Furthermore, as mobile applications commonly exhibit low CPU utilization, we also evaluate the power consumption of each core type with different utilization levels and frequency settings.

A. Performance and Power Behaviors

Although there have been many studies of the architecture and scheduling for asymmetric multi-cores [1]–[3], [5]–[7], commercial asymmetric multi-cores appeared only recently. In this section, we evaluate the performance and power characteristics of real hardware big and little core architectures. In addition to the differences in core microarchitectures, each core type has a different L2 cache size, and in our experiments with the SPECCPU applications, the cache difference affects certain cache-sensitive applications significantly, enlarging the performance gap between the big and little cores. In addition to the core type, the clock frequency of core can change by DVFS, and thus the performance and power trade-offs occur not only across different core types but also different frequency settings. To include the effect of frequency, we evaluate three different frequencies for big cores, 1.9GHz (max), 1.3GHz, and 0.8GHz (min). For the little core, we use the max frequency of 1.3GHz.

For performance comparison, Figure 2 presents the speedup of three big core configurations normalized to the performance with a little core configuration (1.3GHz). Note that only one core is used to run the single-threaded applications. Compared to a little core with the max frequency, a big core exhibits better performance except for the three applications running at the minimum big core frequency. In prior asymmetric core studies, some memory-intensive applications exhibit very low speedups with the same frequency in big and little cores. However, with the difference in the L2 size, at the same frequency, a big core always perform better than a little core for the SPEC applications.

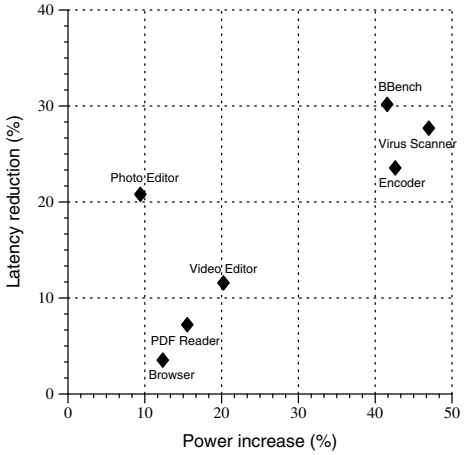

Figure 4. Power and latency for 4 big cores compared to 4 little cores (latency-oriented applications)

The speedup can be up-to 4.5 times with the same 1.3GHz frequency. Figure 3 presents the power consumption in mW. Note that for these experiments with the SPEC applications, the screen and networks are turned off, and the measured power is the full system power. As shown in the figure, at the same frequency, a big core consumes 2.3 times more power than a little core. Even a big core with 0.8GHz consumes 1.5 times more power than a little core with 1.3GHz. The power differences across applications with the same core type and frequency are smaller than the performance differences.

Figure 4 presents performance and power difference results, when 7 latency-oriented mobile applications are running on either 4 little cores or 4 big cores. The y-axis is the latency reduction with big cores, and the x-axis is the power increase with big cores, compared to little cores. Unlike the SPEC applications, the performance difference is relatively small with less than 30% improvement. The difference in power is also much smaller by less than 47% than the SPEC applications. The reason for much smaller performance differences in the mobile applications compared to SPEC is due to their low CPU utilization. Unlike the SPEC applications, the mobile applications do not use cores continuously, and thus, the effect of core architecture is much smaller. Figure 5 presents performance (FPS) and power differences between

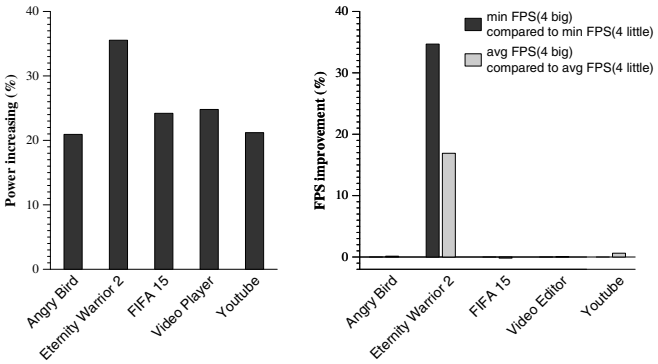

Figure 5. Power and FPS for 4 big cores compared to 4 little cores (FPS-oriented applications)

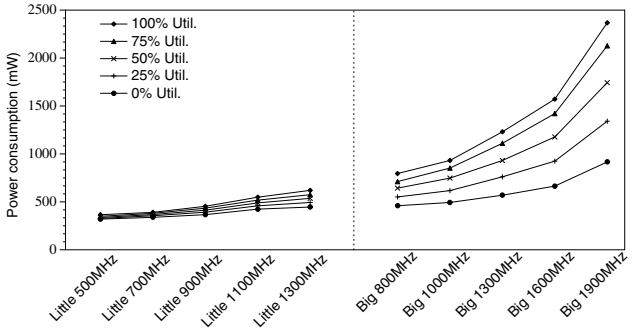

Figure 6. Little and big core power consumption by CPU utilization

big and little cores for 5 FPS-oriented applications. The right-side FPS improvement result shows the FPS improvements for the minimum FPS (dark bar) and the average FPS (light bar). The average FPS difference for each application is much smaller than the latency-oriented ones, except for a CPU-intensive game, `eternity warrior 2`. However, even for the FPS-oriented applications, the worst FPS can be affected by core types, since occasionally computational requirements exceed the limited little core capability, although such occasional slow downs do not change in the average FPS results significantly.

The differences in core architectures as well as cache sizes in the target asymmetric multi-cores provide large performance gaps between big and little cores for the SPECCPU applications, but relatively modest differences for mobile applications. The analysis with the SPECCPU applications only shows the power and performance characteristics when an application utilizes CPUs extensively for a relatively long period of time. However, in mobile platforms, each task becomes active for a very short period, and the CPU utilization is low. The next section will discuss how the CPU utilization affect the power consumption of big and little cores.

B. The Power Effect of Core Utilization

This section presents how the core clock frequency and utilization affect the actual power consumption of the asymmetric processor used in the target platform. For this analysis, we vary two factors. First, core clock frequencies are varied with DVFS from the minimum 500MHz to the maximum 1.3GHz for little cores, and 0.8GHz to 1.9GHz for big cores. Second, we vary the utilization of CPUs by forcing the micro-benchmark to pause periodically to control the CPU utilization. Figure 6 presents the power consumption of big and little cores, with a single core configuration. First, as the utilization increases, the power increase is much higher with high clock frequencies than with low clock frequencies. The system power is much more sensitive to utilization when the clock frequency of cores is high. Second, big and little cores cover clearly different ranges of power for the same utilization.

IV. Scheduling and Power Management

Scheduling and power management affect how application tasks are mapped to asymmetric cores significantly. Core frequency as well as core asymmetry incurs the performance-energy trade-offs. This section discusses the existing scheduler and DVFS governor designs for asymmetric multi-cores.

A. Scheduling Schemes for Asymmetry

There have been three general approaches to map threads to multiple types of cores in prior studies for asymmetric multi-cores. The first approach, *efficiency-based scheduling*, uses the performance efficiency (big core efficiency) of threads, which is the speedup of running an application on a big core over a little core. As each application thread has a different big core efficiency, the most optimal core type is selected based on the performance gain. The second approach, *parallelism-aware scheduling*, is based on available parallelism. When there is an abundant parallelism in an application, more small cores are used, but when the parallelism is low, a big core is used to reduce the length of the critical path. The final approach, *utilization-based scheduling*, considers only the CPU utilization of threads, and schedules threads to big cores, if the threads have high CPU utilization. The last approach is what the current commercial platforms employ.

In efficiency-based scheduling, to maximize throughput, the top N threads with high speedups with big cores are scheduled to N big cores, when N is the number of big cores [1], [2]. If the goal is to save energy, the speedup and power consumption of each type are combined as the efficiency metric. In this type of efficiency-based scheduling, a common requirement is to measure or estimate the big core speedup for each thread for scheduling. Such speedup can be measured by running threads in both cores periodically [1], or can be estimated from instruction throughput or last-level cache misses [5], [6]. Parallelism-aware scheduling aims to exploit the energy efficiency of many small cores, while the performance critical serial phase uses a big core. The scheduler may map applications with less parallelism

Algorithm 1 HMP Scheduler

$Weight = [w_0, w_1, ..., w_N]$
$Runnable\_time\_history = [t_0, t_1, ..., t_N]$

for each task **do**
  $task\_load \leftarrow Weight \cdot Runnable\_time\_history^T$
  if $task\_load >$ HMP_UP_THRESHOLD
    then Little $\rightarrow$ Big migration
  if $task\_load <$ HMP_DOWN_THRESHOLD
    then Big $\rightarrow$ Little migration
end for

Algorithm 2 Interactive governor

for every sampling rate **do**
  $util \leftarrow$ current utilization since last check
  $freq \leftarrow$ current frequency since last check

  $target\_freq \leftarrow freq$ * $util$ / $TARGET\_LOAD$
  if $util >$ UP_THRESHOLD
    if $freq <$ HISPEED_FREQ
      then set frequency to HISPEED_FREQ
    else
      then set frequency to $target\_freq$

  if $util <$ DOWN_THRESHOLD
    then set frequency to $target\_freq$
end for

to big cores, while mapping applications with abundant parallelism to small cores [8]. Parallelism and efficiency can be combined to schedule mixed applications with multi-threaded and single-threaded applications [8].

The aforementioned two approaches in academic studies assume that applications are CPU-intensive and utilize CPU resources continuously. However, in real mobile platforms, CPU loads are relatively low and fluctuate significantly. Considering the fluctuating low utilization of interactive systems, the scheduling approach in practical systems is based on CPU utilization only. If the CPU load of a thread is high beyond the capability of a little core, then, the thread is migrated to a big core. In this utilization-based scheduling, the big core speedup does not need to be measured or estimated since the migration decision is purely dependent upon CPU loads. For systems with low CPU utilization levels, this simple utilization-based scheduling can exploit the performance difference between core types effectively, although there may be some room for improvement by considering accurate big core efficiency and parallelism. The DVFS governor also traces the CPU utilization of each core, and adjusts the frequency to maintain a certain level of CPU utilization by increasing or decreasing CPU frequencies.

B. HMP Scheduler

The HMP scheduler implemented in the target system uses CPU loads to determine a core type for each task. The scheduler tracks the weighted average of CPU load of each task, and if the CPU load exceeds the up-threshold, the thread is migrated to a big core unless the thread already runs on a big core. If the CPU load becomes lower than the down-threshold, it is migrated to a little core. The CPU load is periodically updated. If a task enters the sleep state, its load is not updated.

Algorithm 1 describes the HMP scheduler used in this paper. For every scheduling tick, the CPU load of each task is updated by time-weighted adjustment. The final CPU load is the average of load history weighted differently at one millisecond granularity. The values in the past load history at millisecond unit are weighted by time to give higher weights to more recent past values. In the current implementation, the 1ms-period load generated 32ms ago will be weighted by 50%. Note that the CPU load should be normalized by the current clock frequency, since the scheduler requires an absolute load value independent from the current clock frequency.

Once all the CPU loads are updated, the scheduler checks whether a task on a big core has a low CPU load, or a task on a little core has a high CPU load. If one of the cases occurs, the task migrates to a different type of core. For the core selection decision by the HMP scheduler, three parameters are important for the behavior of the scheduler. The first two parameters are the *up* and *down-thresholds*, to determine when a thread must migrate. The third parameter is the *time weight* to adjust the effect of load history, which determines how much effect each unit of past load values can have for the final scheduling decision. The scheduler also performs traditional load balancing across the same type of cores.

C. CPU Frequency Governor

Once threads are scheduled to cores by migration and load balancing, the frequency of each core is determined by the CPU frequency governor. The most commonly used governor is the `interactive` governor based on CPU loads. Algorithm 2 presents the simplified description of the CPU frequency governor. At every sampling period, which can be configured, and set to 20ms in our system by default, the current CPU utilization of each core is determined. Based on the CPU utilization of each core, the next target frequency is determined to provide enough computing capability to process the CPU load with a small amount of margin for interactivity improvement. One optimization to improve responsiveness of the system is that once the CPU utilization exceed a threshold (`highspeed_freq`), the next frequency is changed to a preset high frequency directly.

One of the most critical parameters for the governor is the *sampling rate*. In the current governor, the CPU load only from the immediately previous period is used to predict the CPU load of the next period. The length of the sampling interval directly affects the past load value. The frequency governor traces the load of each physical core after the scheduling decision is made by the HMP scheduler, working independently from the scheduler.

D. Related Work

For DVFS management based on CPU loads, there have been several early studies with simulation or real implementation [9]–[12]. Weiser et al. explored frequency scaling algorithms with simulation, and the proposed PAST is a precursor to the current interactive governor, which determines the current frequency based only on the CPU load of the previous interval. Grunwald et al. evaluated several DVFS algorithms with real implementation [11]. Flautner et al. proposed to classify tasks to interactive and periodic ones to predict CPU loads based on task execution patterns [10].

There have been prior work for energy saving with the DVFS mechanism for CPU-oriented applications [13]–[17]. These studies attempted to maximize the energy efficiency or the throughput under a certain power constraint. For DVFS-based power management mechanisms of interactive desktop applications, Zhao et al. distinguished interactive tasks from background tasks, and attempted to maintain user-perceived latency under a human-perceptual threshold [18]. Bi et al. also reduces power consumption without increasing perceived latency, utilizing a fine-grained interaction history [19]. Zhu et al. proposed event-based scheduling based on energy-efficient QoS for mobile applications. [7]

There have been several recent studies for analyzing smartphone applications [4], [20]–[23]. As one of the most important smartphone applications, Huang et al. and Wang et al. characterized web-browsing applications [20], [21]. They developed a methodology to generate user inputs automatically for deterministic analysis [20], [21]. Issa et al. and Gutierrez et al. proposed a benchmark suite for smartphones, and characterized their system and architectural behaviors [4], [22]. To find representative system behaviors of smartphones, the smartphone usage patterns haven been studied [24]–[26]. These studies log system activities to investigate the real world usage patterns. Blake et al. and Flautner et al. studied the available TLP for interactive applications [27], [28]. They analyzed the TLP behaviors for common desktop applications. We share a similar method to investigate the TLP of smartphone applications. In parallel with our study, Gao et al. also analyzed TLP behavior for interactive mobile applications on an asymmetric multi-core platform [23]. However, our study further analyzes the effect of scheduler and DVFS governor.

V. Application Characterization

This section presents the thread-level parallelism available in the selected mobile applications on the Android platform. TLP is further decomposed to the little and big core TLP determined by both application characteristics and HMP scheduling.

A. TLP in Mobile Applications

In this section, we first discuss the available TLP and utilization of cores for the applications. Table III presents how little and big cores are used, and how many cores are active during the execution of each application. The two columns, `little` and `big`, show the percentage of cycles when little or big cores are active. The percentage is only from active cycles, excluding the idle cycles which are shown in the `idle` column. The `little` column represents the portion of active cycles, when only little cores are active, and the `big` column represents cycle ratios when at least one big core is active.

App Name	Idle	Little	Big	TLP
PDF Reader	16.14	86.94	13.05	2.06
Video Editor	19.44	89.55	10.44	2.25
Photo Editor	9.06	92.49	7.50	1.40
BBench	0.10	52.16	47.83	3.95
Virus Scanner	2.93	77.25	22.74	2.44
Browser	52.94	94.58	5.41	1.86
Encoder	0.55	37.80	62.19	1.78
Angry Brid	4.41	99.88	0.11	2.34
Eternity Warrior 2	3.65	72.64	27.35	2.85
FIFA 15	9.27	85.62	14.37	2.37
Video Player	14.22	99.38	0.61	2.29
Youtube	12.72	99.92	0.07	2.29

Table III
Thread-level parallelism with 8 cores

Although asymmetric cores consist of four big cores, the actual usage of big cores is significantly lower than expectation. With many applications, except for `bbench`, and `encoder`, for 72-99% of the active cycles, big cores are not used at all. Among the benchmark applications, `virus scanner`, `encoder`, `bbench`, and `eternity warrior` exhibit high 22-62% active cycles with big cores.

The last column, `TLP`, shows the average number of active cores (both little and bit cores) when the processor is not completely idle. We use the same metric as proposed by Blake et al. for TLP [27]. The TLP metric represents how many cores are active on average during the non-idle execution cycles. The overall TLP is very low for mobile interactive applications. All of the benchmark applications, except for `bbench`, have less than 3 cores active on average. This result indicates the current 8 core design has a significant amount of idle CPU resource.

Although such low TLPs have been expected for mobile applications, the measured results confirmed the severity of the lack of parallelism. Compared to desktop ones, mobile applications have a limited screen interface, which further restricts the number of simultaneously active applications. Even though the current multi-cores have 8 cores for mobile systems, only two or three of them are used for the majority of applications.

B. TLP by Core Type

In this section, we decompose the execution cycles by big and small core TLPs. Table IV presents the percentage of execution cycles decomposed by the number of active little and big cores. Each row represents the number of active

PDF Reader		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	16.14	33.41	15.56	9.46	4.10
	C1	1.31	6.84	6.09	4.07	1.75
	C2	0.03	0.31	0.23	0.36	0.20
	C3	0.00	0.01	0.01	0.03	0.00
	C4	0.00	0.00	0.00	0.00	0.00

Video Editor		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	19.44	26.05	19.20	12.23	11.00
	C1	1.81	1.95	1.47	1.74	1.02
	C2	1.20	0.39	0.17	0.12	0.17
	C3	0.59	0.34	0.05	0.05	0.00
	C4	0.41	0.25	0.14	0.05	0.05

Photo Editor		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	9.06	64.81	17.25	4.01	0.94
	C1	0.35	0.27	0.23	0.09	0.13
	C2	0.63	0.19	0.01	0.00	0.00
	C3	0.69	0.21	0.01	0.01	0.00
	C4	0.51	0.33	0.09	0.01	0.00

BBench		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	0.10	0.33	0.83	1.08	0.71
	C1	0.92	6.47	8.67	6.78	5.17
	C2	6.51	13.26	12.99	8.98	6.18
	C3	2.28	4.65	5.09	3.81	2.93
	C4	0.37	0.52	0.54	0.44	0.27

Virus Scanner		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	2.93	13.34	20.09	17.52	10.55
	C1	10.35	5.27	3.67	2.64	1.23
	C2	4.20	2.08	0.72	0.38	0.24
	C3	1.39	1.29	0.36	0.16	0.04
	C4	0.56	0.50	0.26	0.10	0.02

Browser		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	52.94	23.16	10.97	4.94	3.52
	C1	0.65	0.94	1.05	0.94	0.55
	C2	0.00	0.11	0.03	0.09	0.03
	C3	0.00	0.00	0.00	0.00	0.00
	C4	0.00	0.00	0.00	0.00	0.00

Encoder		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	0.55	0.39	0.28	0.20	0.19
	C1	47.34	27.76	9.47	2.82	1.19
	C2	5.01	2.13	0.41	0.15	0.09
	C3	0.83	0.52	0.03	0.03	0.00
	C4	0.21	0.24	0.03	0.01	0.00

Angry Bird		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	4.41	21.16	33.91	26.50	13.75
	C1	0.01	0.09	0.01	0.05	0.05
	C2	0.00	0.00	0.00	0.00	0.00
	C3	0.00	0.00	0.00	0.00	0.00
	C4	0.00	0.00	0.00	0.00	0.00

Eternity Warrior 2		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	3.65	8.28	8.88	7.71	5.68
	C1	8.84	13.78	13.91	11.11	8.84
	C2	1.18	2.28	2.69	1.76	1.04
	C3	0.03	0.06	0.08	0.05	0.03
	C4	0.00	0.00	0.00	0.00	0.00

FIFA 15		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	9.27	20.23	21.11	12.98	7.97
	C1	3.59	7.57	7.48	4.49	2.79
	C2	0.50	0.62	0.61	0.39	0.20
	C3	0.02	0.02	0.04	0.01	0.00
	C4	0.00	0.00	0.00	0.00	0.00

Video Player		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	14.22	24.17	26.09	19.89	14.55
	C1	0.21	0.25	0.30	0.02	0.07
	C2	0.01	0.04	0.04	0.01	0.05
	C3	0.00	0.00	0.00	0.00	0.00
	C4	0.00	0.00	0.00	0.00	0.00

Youtube		Little Cores				
		C0	C1	C2	C3	C4
Big Cores	C0	12.72	27.20	23.39	20.34	16.18
	C1	0.00	0.03	0.03	0.09	0.00
	C2	0.00	0.00	0.00	0.00	0.00
	C3	0.00	0.00	0.00	0.00	0.00
	C4	0.00	0.00	0.00	0.00	0.00

Table IV
TLP DISTRIBUTIONS BY CORE TYPE

big cores from 0 to 4 cores, and each column represents the number of active little cores. For example, row-C0 and column-C0 is the cycle percentage with none of the 8 cores are actively executing any task. To measure this result, the CPU states are checked at every 10ms. However, this result does not show the actual utilization of core during each 10ms period, presenting only how many cores have a non-zero utilization during each sampling interval.

In general, TLP in little cores is much higher than TLP in big cores. The majority of available TLP is processed by energy-efficient little cores, due to the low CPU loads in mobile applications. Even if big cores are used, only one of the cores is used most of the time. For a short burst of CPU load increase in the four applications (eternity warrior, bbench, virus scanner, encoder), a single big core can absorb the load demand very effectively. Considering this usage pattern and TLP for big and little cores, the current design with four big and four small cores may not be optimal asymmetric multi-cores for mobile systems.

C. Performance and Power with Different Asymmetric Core Combinations

To analyze the effect of different little and big core combinations, we measure performance and power with 7 different combinations of little and big cores, from only two little cores, to 4 little cores with two big cores. Figures 7

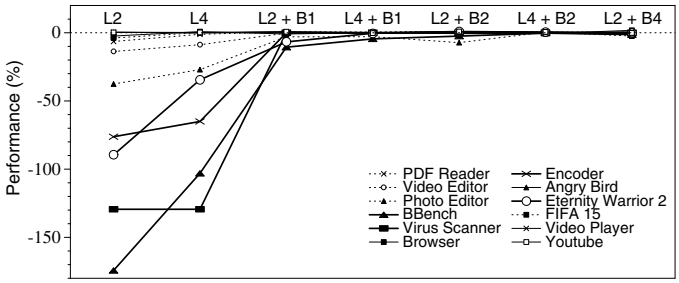

Figure 7. Performance (latency and FPS) with various core configurations

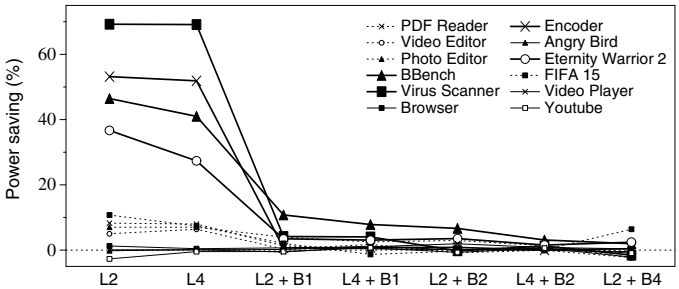

Figure 8. Power saving with various core configurations

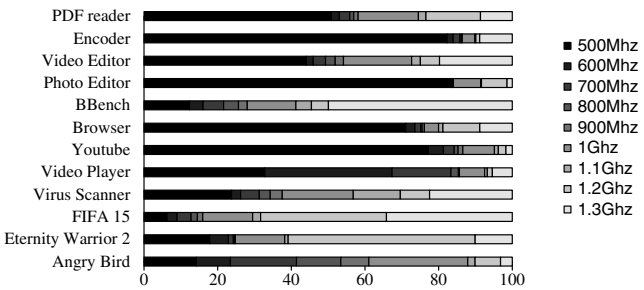

Figure 9. Little core frequency distribution

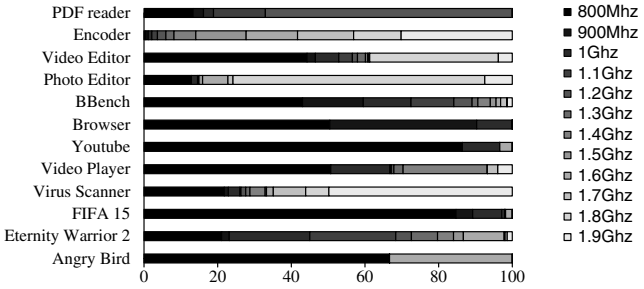

Figure 10. Big core frequency distribution

and 8 present performance and power changes with the 7 big and little core configurations. For example, L2+B4 denotes a configuration where two little cores and four big cores are enabled. The power saving and performance improvement are compared to the baseline with four little and four big cores enabled. Note that since these examined configurations always have less cores than the baseline, the power consumption and performance do not exceed those of the baseline.

The result indicates that having only little cores can reduce power consumption significantly. For a subset of applications, such as angry bird and video player, having only little cores can gain power saving without performance degradation. However, the performance reduction is very severe for the other applications. Although big cores are used infrequently, having a single big core has a significant impact by supporting fast interactivity of mobile applications. Among the seven configurations, two little cores with one big core, or four little cores with one big core provide a good performance-power balance.

VI. The Effect of Scheduling and DVFS

With core asymmetry, the HMP scheduler and CPU frequency governor have a significant impact on the performance and energy behavior of mobile asymmetric platforms. In this section, we investigate how the two system components affect the behavior of mobile systems.

A. Core Frequency Changes

Based on fluctuating CPU loads, the frequency governor changes core frequencies at every sampling period. 20ms sampling period is the default setting in the our platform. In this section, we first analyze what frequency the governor sets for little and big cores when our benchmark applications run. Figure 9 and 10 present the CPU frequency decomposition for little and big cores. The distribution only includes active periods for each core, ignoring idle cycles.

App Name	State					
	Min	$< 50\%$	$< 70\%$	70-95%	$> 95\%$	Full
PDF Reader	44.12	41.05	5.33	5.16	4.32	0.00
Video Editor	36.86	45.56	7.31	6.32	3.89	0.04
Photo Editor	77.06	19.45	1.23	0.98	1.15	0.10
BBench	6.83	40.96	14.47	13.14	24.45	0.12
Virus Scanner	18.02	49.26	17.22	9.00	3.61	2.85
Browser	67.70	19.80	3.35	4.48	4.66	0.00
Encoder	53.05	20.63	0.99	1.35	20.00	3.94
Angry Bird	13.27	68.54	7.76	8.40	2.01	0.00
Eternity Warrior 2	11.96	64.11	9.07	8.56	6.10	0.16
FIFA 15	4.39	69.12	14.56	6.54	5.36	0.00
Video Player	27.98	47.60	11.91	9.82	2.65	0.01
Youtube	73.63	17.42	3.05	4.24	1.63	0.00

Table V
Efficiency Decomposition

Big and little cores show slight different distributions in the figures. For little cores as shown in Figure 9, more diverse patterns of frequency distribution appear for our applications. For example, `video player` has very low core utilization, and thus the lowest frequency dominates the distribution. `Eternity warrior` has a relatively high fluctuating core utilization, and thus exhibits a wide variety of core frequencies. For big cores as shown in Figure 10, several latency-oriented workloads such as `encoder`, `photo editor`, and `virus scanner` use big cores intensively to absorb burst loads, and big core frequencies are set to high clock frequencies. However, for games, browsers, and video players with relatively steady loads, big cores are used mostly at low frequencies to handle extra CPU loads occasionally which exceed the little core capacity.

B. Scheduler and Governor Efficiency

To further analyze the effect of scheduler and governor, we investigate how effectively the scheduler and governor set the core type or frequency. Since both schemes are based on CPU loads, the ideal scheduler and governor should find the optimal core type and frequency which maximize the CPU utilization at the given core type and frequency setting. Assigning a big core or setting a high frequency for a task with low utilization, will waste the energy budget. However, if the setting of core type and frequency is too low to accommodate the current CPU load, the performance or interactivity of applications will be degraded. Ideally, the scheduler and governor must maintain the CPU utilization close to 100%, which matches the exact CPU requirement of the task, by selecting the best core type and frequency. In practice, however, the system leaves some margin to utilization for unpredicted increases of CPU loads.

To quantify the effect of scheduler and governor, we analyze how the CPU utilization is maintained by the scheduler and governor. For the utilization analysis, we decompose the execution cycles into six categories. Note that we measure the utilization at every 10ms. The first category, `full`, is the state when the current CPU load can fully utilize the full CPU capacity with a big core at its maximum frequency. In this state, although the scheduler and governor set the core and frequency with correct prediction, the load exceeds the maximum CPU capacity. The second category, `>95%`, is the state when the current load requires more CPU capacity than the current core type or frequency with more than 95% utilization, since the utilization cannot exceed 100%. Based on the utilization, we decompose the state into 70-95%, 50-70%, and less than 50%. The final category, `min`, is the state when the CPU load is less than 50%, but the current core is a little core with the minimum frequency. In this case, even if the load is low, the scheduler and governor cannot further reduce CPU capability due to the hardware limit. Note that except for `min` and `full`, the utilization results do not consider the clock frequency and core type, since this analysis investigates how much CPU capacity is wasted due to the inaccurate load prediction.

Table V presents the cycle decomposition with the six categories for our benchmark applications. For the baseline HMP scheduler and governor, surprisingly, the majority of cycles are either in `min` or `<50%` state. The `min` state occurs since the current DVFS cannot decrease the frequency lower than 500MHz, and for many applications, they require less computing capability than a 500MHz little core for a quite significant portion of their execution times. Based on the observation, to further reduce the energy consumption, another core type, tiny core, with much weaker capability can be added to process such low CPU loads.

Since the scheduler and governor maintain a utility margin to ensure fast response combined with their low accuracy in predicting next loads, the majority of execution times are spent in `<50%` state. With a more aggressive scaling with more accurate load prediction, the power can be further reduced compared to the current conservative design of governors. For `bbench`, and `encoder`, the loads increase in a burst manner, so there are high ratios of `>95%`, when the DVFS does not increase the clock frequency fast enough. For `virus` and `encoder`, 2-4% of the cycles are in `full` state when the big core is fully utilized at the highest frequency.

C. Scheduler and Governor Effect

To investigate the effect of scheduler and governor, we use 8 different configurations of the HMP scheduler and frequency governor in addition to the default baseline setting. For the governor effect, we first increase the sample interval of frequency governor to 60 or 100ms from the default 20ms. Increasing the sample interval has a trade-off. With a longer interval, the governor may not respond to fast changes of CPU loads quickly, and thus lead to the degradation of system interactivity. However, with the longer interval, the CPU load can be measured at more stable state than with the shorter interval. With the stable utilization, the next CPU load can be more predictable, allowing accurate frequency setting. The next two configurations, `high (80)` and `low (60)` target loads, adjust the threshold jumping to high frequency setting in the interactive governor, changing how quickly the clock frequency jumps to the high level. The default target load is 70 in our platform.

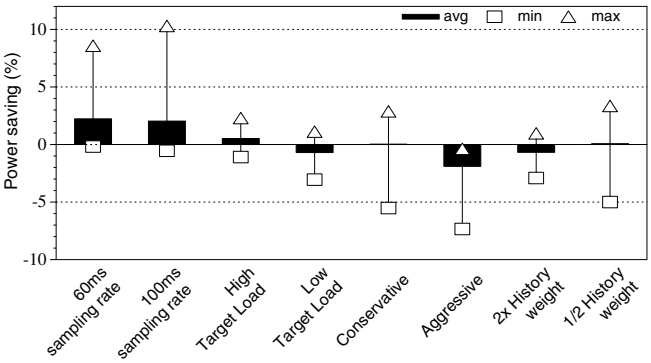

Figure 11. Power saving with 8 governor/HMP parameters (all applications)

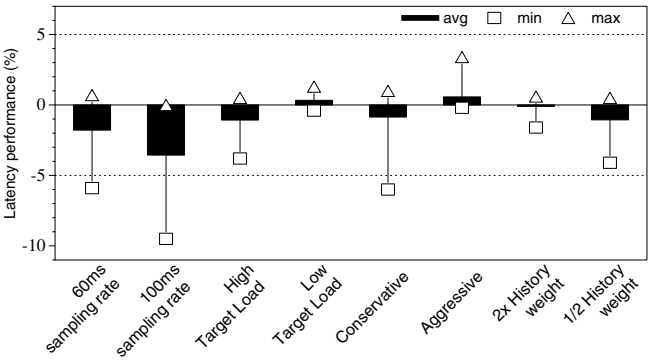

Figure 12. Latency changes with 8 governor/HMP parameters (latency-oriented applications)

The next four configurations change the HMP scheduler. The first HMP configuration, `conservative` (850,400), forces the HMP scheduler to schedule to little cores more eagerly, and the second HMP configuration, `aggressive` (550,100), forces the HMP scheduler to schedule tasks to big cores more eagerly by decreasing the up-threshold and down-threshold. The default up and down threshold values are 700 and 256, respectively. The next two HMP configurations (`2x history weight` and `1/2 history weight`) change the weight scale of load history to double or half from 32 in the baseline. The weight scale changes affect the weights in load history, either giving more or less weights to recent loads.

Figure 11 presents the power saving with 8 different configurations compared to the baseline system. Each bar

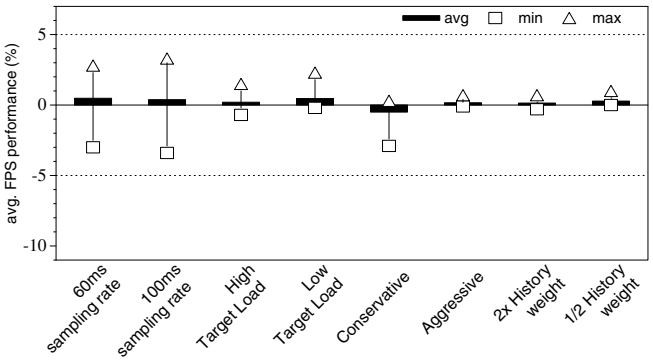

Figure 13. Avg. FPS changes with 8 governor/HMP parameters (FPS-oriented applications)

shows the average power saving for the corresponding configuration, and the min-max range is also shown in the figure. Figures 12 and 13 show performance changes in latency and FPS with the 8 configurations. In the figures, the left side four bars present the configurations for DVFS policies, and the right four bars show the HMP scheduler changes. Among the 8 configurations, the sampling interval has a relatively high impact on the power saving with up-to 10% for `bbench`. Although increasing the sampling interval to 60ms reduces power by 2% on average, but higher for some applications, the performances tend to decrease, especially affecting the latency of some applications. Therefore, the interval change must be used carefully not to degrade the responsiveness of applications. On average, the rest of configurations do not exhibit significant changes. However, the min-max ranges are sensitive to certain configuration changes. The `conservative` HMP configuration can reduce the power for certain applications, while increasing the latency for the worst case application. Similarly, the `aggressive` HMP configuration mostly increases the power consumption (negative power saving). However, the weight scale has only a minor impact on the overall power saving.

Based on the results with the 8 different DVFS and HMP scheduler parameters, the most critical one is the sampling window interval for the DVFS governor, with a modest but noticeable trade-off between power and performance. The current interactive governor predicts the future load only based on the immediately near past, and CPU loads change very fast in mobile applications. Therefore, selecting a right interval for future load prediction is critical for the DVFS governor. The parameters of the HMP scheduler have a relatively minor impact, as CPU loads for big cores tend to have a bi-modal distribution. When a big core is used, the application has a high load, and thus the scheduler will migrate the thread to a big core regardless of minor changes in scheduler parameters.

VII. Conclusion

This paper characterized three aspects of mobile asymmetric platforms, core architecture, application, and scheduler. Although the current asymmetric architecture provides clear performance and energy trade-offs envisioned by prior studies, the quantification of TLP and usage patterns of mobile applications confirmed that the current asymmetric multi-cores are unnecessarily over-designed to have more computing capability than mobile systems demand. For a significant fraction of execution cycles, even the little core with the minimum frequency has too much computing capability. However, having at least one big core is critical for the latency and the worst case FPS of many mobile applications, even if the big core is used much less frequently than little cores. A close investigation of the efficiency of governor and scheduler revealed that frequency and core types are often conservatively set by the scheduler and governor, not to degrade the performance, also due to the limitation of the current load prediction scheme.

Acknowledgment

This research was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIP) (No. NRF-2012R1A1A1014586 and No. NRF-2013R1A2A2A01015514).

References

[1] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2003, p. 81.

[2] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance," in *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA)*, 2004, p. 64.

[3] J. C. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar, "Using asymmetric single-ISA CMPs to save energy on operating systems," *IEEE Micro*, vol. 28, no. 3, pp. 26–41, 2008.

[4] A. Gutierrez, R. Dreslinski, T. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *2011 IEEE International Symposium on Workload Characterization (IISWC)*, 2011, pp. 81 –90.

[5] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "HASS: a scheduler for heterogeneous multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 66–75, 2009.

[6] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multi-core architectures," in *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*, 2010, pp. 125–138.

[7] Y. Zhu, M. Halpern, and V. Reddi, "Event-based scheduling for energy-efficient qos (eqos) in mobile web applications," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 137–149.

[8] J. C. Saez, M. Prieto, A. Fedorova, and S. Blagodurov, "A comprehensive scheduler for asymmetric multicore systems," in *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*, 2010, pp. 139–152.

[9] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 1994.

[10] K. Flautner, S. Reinhardt, and T. Mudge, "Automatic performance setting for dynamic voltage scaling," *Wirel. Netw.*, vol. 8, no. 5, pp. 507–520, Sep. 2002.

[11] D. Grunwald, C. B. Morrey, III, P. Levis, M. Neufeld, and K. I. Farkas, "Policies for dynamic clock scheduling," in *Proceedings of the 4th Conference on Symposium on Operating System Design and Implementation (OSDI)*, 2000, pp. 6–6.

[12] K. Govil, E. Chan, and H. Wasserman, "Comparing algorithm for dynamic speed-setting of a low-power cpu," in *Proceedings of the 1st Annual International Conference on Mobile Computing and Networking (MobiCom)*, 1995, pp. 13–25.

[13] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2006, pp. 347–358.

[14] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Proceedings of the 2004 international symposium on Low power electronics and design (ISLPED)*, 2004, pp. 174–179.

[15] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser, "Koala: a platform for os-level power management," in *Proceedings of the 4th ACM European conference on Computer systems (EuroSys)*, 2009, pp. 289–302.

[16] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable power control for many-core architectures running multi-threaded applications," in *Proceedings of the 38th annual international symposium on Computer architecture (ISCA)*, 2011, pp. 449–460.

[17] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack and Cap: adaptive dvfs and thread packing under power caps," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 175–185.

[18] X. Zhao, Y. Guo, and X. Chen, "Transaction-based adaptive dynamic voltage scaling for interactive applications," in *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design (ISLPED)*, 2009, pp. 255–260.

[19] M. Bi, I. Crk, and C. Gniady, "IADVS: On-demand performance for interactive applications," in *IEEE 16th International Symposium on High Performance Computer Architecture (HPCA)*, 2010, pp. 1 –10.

[20] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl, "Anatomizing application performance differences on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys)*, 2010, pp. 165–178.

[21] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie, "Why are web browsers slow on smartphones?" in *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications (HotMobile)*, 2011, pp. 91–96.

[22] J. Issa, Q. Le, S. Min, J. Steinbrecher, and R. Turlapati, "TMAPP - typical mobile applications benchmark," in *proceedings of the Seventh Annual Workshop on Modeling, Benchmarking and Simulation (MoBS)*, 2011.

[23] C. Gao, A. Gutierrez, M. Rajan, R. Dreslinski, T. Mudge, and C.-J. Wu, "A study of mobile device utilization," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*, March 2015, pp. 225–234.

[24] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," in *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys)*, 2010, pp. 179–194.

[25] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 168–178.

[26] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman, "Identifying diverse usage behaviors of smartphone apps," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference (IMC)*, 2011, pp. 329–344.

[27] G. Blake, R. G. Dreslinski, T. Mudge, and K. Flautner, "Evolution of thread-level parallelism in desktop applications," in *Proceedings of the 37th annual international symposium on Computer architecture (ISCA)*, 2010, pp. 302–313.

[28] K. Flautner, R. Uhlig, S. Reinhardt, and T. Mudge, "Thread-level parallelism and interactive performance of desktop applications," in *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000, pp. 129–138.