A Study of Core Utilization and Residency in Heterogeneous Smart Phone Architectures

Joseph Whitehouse
whitehouse@utexas.edu
University of Texas at Austin
Austin, Texas

Qinzhe Wu
qw2699@utexas.edu
University of Texas at Austin
Austin, Texas

Shuang Song
songshuang1990@utexas.edu
University of Texas at Austin
Austin, Texas

Eugene John
eugene.john@utsa.edu
University of Texas at San Antonio
San Antonio, Texas

Andreas Gerstlauer
gerstl@ece.utexas.edu
University of Texas at Austin
Austin, Texas

Lizy K. John
ljohn@ece.utexas.edu
University of Texas at Austin
Austin, Texas

ABSTRACT

In recent years, the smart phone platform has seen a rise in the number of cores and the use of heterogeneous clusters as in the Qualcomm Snapdragon, Apple A10 and the Samsung Exynos processors. This paper attempts to understand characteristics of mobile workloads, with measurements on heterogeneous multicore phone platforms with big and little cores. It answers questions such as the following: (i) Do smart phones need multiple cores of different types (eg: big or little)? (ii) Is it energy-efficient to operate with more cores (with less time) or fewer cores even if it might take longer? (iii) What are the best frequencies to operate the cores considering energy efficiency? (iv) Do mobile applications need out-of-order speculative execution cores with complex branch prediction? (v) Is IPC a good performance indicator for early design tradeoff evaluation while working on mobile processor design?

Using Geekbench and more than 3 dozen Android applications, and the Workload Automation tool from ARM, we measure core utilization, frequency residencies, and energy efficiency characteristics on two leading edge smart phones. Many characteristics of smartphone platforms are presented, and architectural implications of the observations as well as design considerations for future mobile processors are discussed. A key insight is that multiple big and complex cores are beneficial both from a performance as well as an energy point of view in certain scenarios. It is seen that 4 big cores are utilized during application launch and update phases of applications. Similarly, reboot using all 4 cores at maximum performance provides latency advantages. However, it consumes higher power and energy, and reboot with 2 cores was seen to be more energy efficient than reboot with 1 or 4 cores. Furthermore, inaccurate branch prediction is seen to result in up to 40% mis-speculated instructions in many applications, suggesting that it is important to improve the accuracy of branch predictors in mobile processors. While absolute IPCs are observed to be a poor predictor of benchmark scores, relative IPCs are useful for estimating the impact of microarchitectural changes on benchmark scores.

ICPE '19, April 7–11, 2019, Mumbai, India

ACM ISBN 978-1-4503-6239-9/19/04…$15.00
https://doi.org/10.1145/3297663.3310304

CCS CONCEPTS

• **Computer systems organization** → *Multicore architectures*; *Heterogeneous (hybrid) systems*; Cellular architectures.

KEYWORDS

Measurement, Multicore Utilization, Frequency Residency, Energy Efficiency, Smart Phone CPUs, Workload Characterization.

ACM Reference Format:
Joseph Whitehouse, Qinzhe Wu, Shuang Song, Eugene John, Andreas Gerstlauer, and Lizy K. John. 2019. A Study of Core Utilization and Residency in Heterogeneous Smart Phone Architectures. In *Tenth ACM/SPEC International Conference on Performance Engineering (ICPE '19), April 7–11, 2019, Mumbai, India.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3297663.3310304

1 INTRODUCTION

Smart phones are becoming an increasingly central part of everyday life. Smart phones connect people to social media, business, banking, and more. This dependence on smart phones all-day long has created the desire for a handheld powerful supercomputer with a battery that lasts for days. While this may be an exaggeration, there is pressure to develop more powerful phones with energy-efficient operation. This challenge has been addressed differently throughout the industry. In recent years, the mobile platform has seen a rise in the number of cores and the use of heterogeneous clusters. The idea is to take advantage of different architectures, switching between efficiency and performance. This brings to light many questions involving the number and nature of cores (big, little, tiny, etc.) and useful core configurations.

It is challenging to design energy-efficient processors for emerging smart phones due to the variety of applications and their disparate features and execution characteristics. This paper attempts to understand characteristics of mobile workloads, especially in the context of heterogeneous multicores. In the process, we aim to answer several questions related to state-of-the-art mobile CPU design.

Many present-day phones use heterogeneous multicore processors in them, for instance, the Qualcomm Snapdragon 810, and

the Samsung ExynosSoCs have 4 big and 4 little ARM cores. The Apple A10 Fusion core has 2 high performance cores and 2 energy-efficient cores. A recent ISCA Keynote Speech [18] alluded to the low utilization of many multicore processors.

Past research from Gao *et al.* [13] and Seo *et al.* [19] argued for fewer cores or tinier cores based on the thread-level parallelism (TLP) inherent to popular applications and the power consumption of different applications. We present CPU usage analysis for several popular applications on the two leading edge smart phones and explore optimal core counts and core types. We specifically study whether big cores yield significant performance improvements compared to littles ones, how much more computation capabilities big cores have, and what applications give the best and what applications the least differential over the little cores. Furthermore, do smart phone applications need multiple cores? Do they need multiple big and little cores? And what are their best operating frequencies from the energy efficiency point of view? Low-power operation will result in longer run times. What are the tradeoffs from the perspective of minimizing energy, energy-delay (ED) or energy-delay^2(ED^2)?

In the server world, it is frequently told that, for energy-efficient operation, the best policy is to "race to idle", i.e. run at the highest frequency (and highest performance/power) to finish the task as soon as possible and then switch to inactive modes. In the mobile multicore context, is it more energy-efficient to use as many big cores simultaneously and race to finish or is it better to use fewer or little cores and reduce power? We explore whether the race-to-idle policy is applicable for mobile workloads on multicore smart phones.

Within this context, an interesting question is how complex the big or how simple the little cores should be. More specifically, do mobile applications need an out-of-order speculative execution core with a sophisticated branch predictor? Many mobile applications are known to be more loop-intensive than benchmarks such as SPEC CPU-INT and it is generally thought that loop branches are easy to predict. Are there lots of branch mispredictions in mobile workloads? How does branch misprediction affect a mobile platform? In many desktop/server workloads, 25-30% instructions are in the mis-speculated paths [9]. What percent of instructions are executed in mis-speculated paths in mobile workloads?

To optimize processor micro-architecture characteristics, such as branch prediction, especially in early design tradeoff evaluations, architects often use instructions per cycle (IPC) as an indication of performance. But what is the effect of IPC on final mobile application performance? If IPC is improved, does it improve the actual application performance? In other words, is IPC even a good metric for pre-silicon performance studies of mobile processor micro-architectures?

Using Geekbench, and several Android applications, we study core utilization, frequency residencies, energy efficiency characteristics, and discuss architectural implications of the observations. Utilizing tools such as workload automation, Linux performance monitoring utilities, and hardware performance monitoring counters, we derive insight into the working of a smart phone. Our experiments unveil insights on the complex interactions between operating systems, dynamic voltage and frequency scaling (DVFS), performance requirements of various types of every day applications, resource sharing and resource contention, etc. in the context of actual applications running on an actual phone.

Based on our workload characterization, we argue for the inclusion of big, complex cores for the computation demand in mobile environments. It appears from our experiments that all 4 big cores are fully utilized during application launch and also during updating of applications. Big cores are needed to provide the computing power needed during such phases and possibly during multi-tasking. One may think of application updates as a non-critical operation that can be accomplished when phones are charging. However, in many scenarios, an application may deny you access until the update is finished and it ends up being a critical operation when a user needs to immediately access the application.

A consequence is that even in the mobile world, under many conditions, the race-to-idle philosophy is useful. In our experiments, for application update scenarios, we observe that it is beneficial to simultaneously use all 4 big cores from performance and energy standpoints. As long as big cores are turned off in the phases they are not required, where appropriate design of frequency governors and schedulers become extremely important, availability of multiple big and complex cores in mobile platforms is of benefit.

We also observe that branch mispredictions result in up to 40% wasted (aborted) instructions in 5 of the Geekbench applications. Essentially for every 100 instructions that are retired by the processor up to 140 instructions have to be fetched and decoded. Similar observations have been noted in the past for desktop and server processors [9], but we confirm that they occur in mobile platforms as well. The extra fetches and decodes cost extra power/energy, however do not end up being useful to the overall execution. Based on this we argue for an accurate branch predictor even in mobile applications (which are thought to be loop-intensive).

In the quest for meaningful metrics for such micro-architecture studies, we observe that there is no linear correlation between IPC and Geekbench score. However, when we compute the difference in IPC between the big core and the little core, we noticed that the IPC delta between the 2 types of cores and the Geekbench score delta between the 2 cores have a linear correlation. This indicates that IPC can serve as a useful metric for relative (but not necessarily absolute) comparisons between different micro-architectures in a mobile context.

2 RELATED WORK

Several groups have performed workload characterization of mobile platforms. Gutierrez *et al.* [15] investigated mobile benchmarks and attempted to implement their own suite. They developed an alternative benchmark, Bbench, claiming that SPEC benchmarks are not representative. Bbench is a benchmark suite that measures the performance of web browsing based on page rendering speed [15]. The characterization presented in this paper includes only microarchitectural metrics, but our work includes utilization and frequency residencies on an actual phone platform under dynamic voltage and frequency scaling. Pandiyan [17] expands on Bbench, by adding more realistic web browsing scenarios and including photo and video applications. Gomez [14] presents a tool, RERAN, that allows actions on a phone to be recorded and replayed. Using everyday

Table 1: Workloads used in our experiments.

Category	Workloads
General	Clock, Calendar, Email, Messages, Application Download, Application Update, Skype, Google Hangout
Camera	Take photos, Edit Photos, Browse Photos, Filters/Effects, Panoraic Photo, Record and Playback video, Burst Photos, Animated Photos
Web Browsing	ESPN, News Sites, Wikipedia, Google Images, Facebook
Games	Crossy Road, Clash Royale, Pokemon go, Real boxing, Asphalt8, NOVA 3, Mortal Komat X, Angry Birds, Temple Run
Virtual Reality	Archer E Bowman, Caldera Defense, Cosmos Warfare, Epic Dragon VR, Gunjack, InCell VR, Snake VR, Temple Run VR, Colosse, DreamWorks VR, Next VR
Multiscreen	Skype + Web Browsing, Skype + YouTube, Skype + Gallery, Skype + Facebook, FaceBook + YouTube, Facebook + Gallery
Benchmarks	Geekbench, SysBench (See Table 2 for more details on Geekbench)

Table 2: Geekbench 3 workloads.

Category	Benchmarks
Integer	AES, Twofish, SHA1, SHA2, Bzip2 Compress, Bzip2 Decompress, JPEG Compress, JPEG Decompress, PNG Compress, PNG Decompress, Sobel, Lua, Dijstra
Floating Point	Black-Scholes, Mandelbrot, Sharpen Filter, Blur Filter, SGEMM, DGEMM, SFFT, DFFT, N-Body, RayTrace
Memory	Stream Copy, Stream Scale, Stream Add, Stream Triad

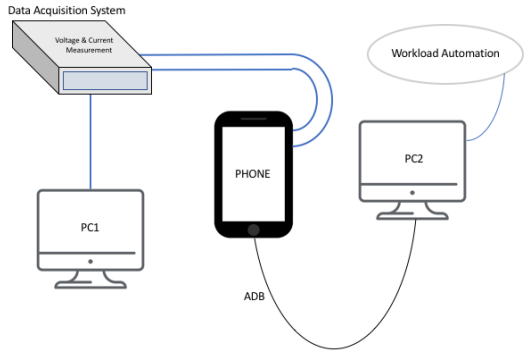


Figure 1: Measurement Methodology

situations is appealing because they are actual end user behavior. Using RERAN allows these situations to be recorded and replayed, which brings repeatability to the day-of-use scenarios. We used RERAN in some of our early experiments, but once the Workload Automation tool [6] became available from ARM, it provided various capabilities needed in our experiments, and we used it for most of our experiments.

A few groups have investigated issues surrounding the use of many cores for embedded and mobile applications [1, 2, 4, 7, 8, 10, 11, 13, 19–21]. Gao *et al.* [13] used RERAN to simulate different popular applications. Using these scenarios, Gao investigated the amount of thread level parallelism (TLP) available, concluding that with the TLP inherent to popular applications, only 2 cores were necessary. Seo *et al.* [19] investigated the available TLP and the impact of core asymmetry on applications in a Samsung Galaxy S5. They also measured power while evaluating different applications and SPEC2006 benchmarks, proposing the need for a tiny core and concluding that a 2 big, 1 little system was equivalent to an octa-core system. Several researchers [7, 8] investigated energy-performance tradeoffs with unique perspectives, introducing inefficiency metrics, optimizing the amount of extra energy that can be allowed for performance, or considering user experience. Zhu *et al.* [21] investigated how the link between the network and CPU affects energy efficiency in mobile web browsing, showing that the CPU only matters when there is low latency in the network. The utilization of multicore smartphone processors and their thermal issues were presented in [1, 2, 4].

3 METHODOLOGY

We analyze the performance-energy tradeoffs in heterogeneous multicore platforms using an Odroid board [5], and two leading-edge smart phone target platforms. The first phone is a circa 2015 phone and we refer to it as Platform 1 in the rest of the paper. The other phone is a circa 2016 phone and we refer to it as Platform 2 in the rest of the paper. All the platforms have heterogeneous processor clusters popularly known as the big.Little platforms, with 4 big cores and 4 little cores. Both the smart phone platforms that we experimented with use quad-core clusters with the ARM ISA and a GPU.

Several benchmarks and applications of everyday use scenarios were investigated. We use everyday applications constituting what are referred to as common daily scenarios. These common daily scenarios are made of tasks that aim to emulate how people would actually use their phone, including email, calendar, messaging, etc.

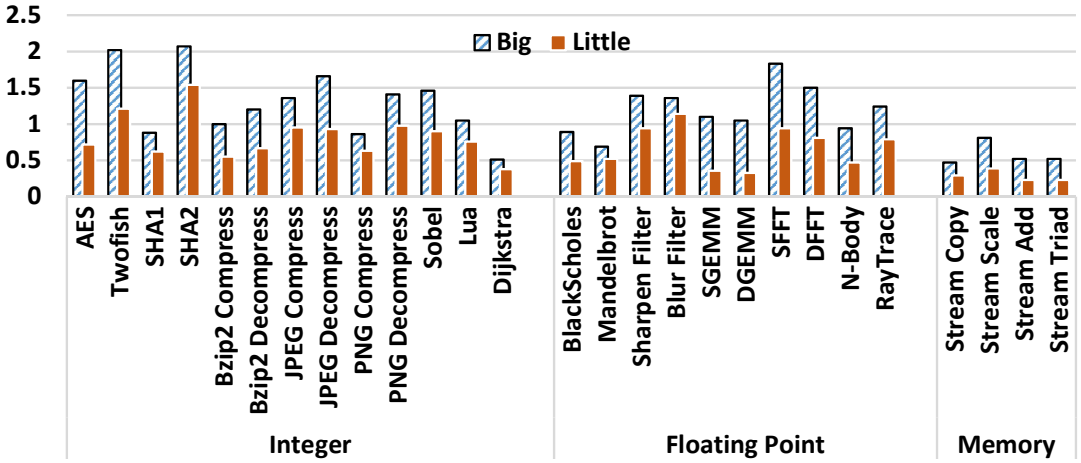
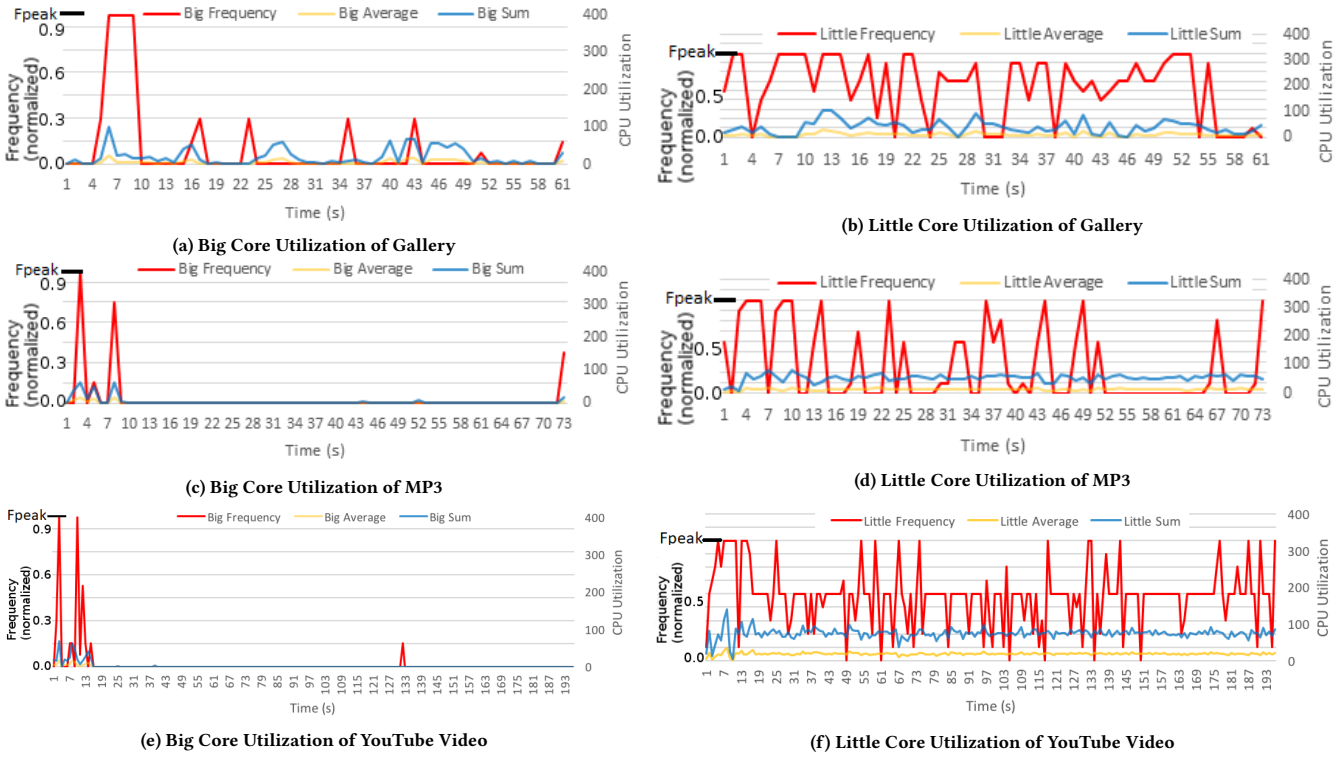


Figure 2: IPC of big core and little core for various Geekbench applications.



(a) Big Core Utilization of Gallery

(b) Little Core Utilization of Gallery

(c) Big Core Utilization of MP3

(d) Little Core Utilization of MP3

(e) Big Core Utilization of YouTube Video

(f) Little Core Utilization of YouTube Video

Figure 3: Core utilization for example workloads in 4-Big, 4-Little configuration. The yellow line shows average per core utilization and the blue line shows sum of the utilization.

Table 1 lists the various scenarios used in our study. In addition to various common daily scenarios, Geekbench [3], a popular benchmark suite and SysBench [16], another benchmark suite were also used. The individual Geekbench 3 workloads are shown in Table 2. By observing benchmarks and common daily scenarios in addition to other created everyday use scenarios such as application launches and updates, a comprehensive picture of the mobile platform is obtained. Several performance counters were chosen to quantitatively evaluate different benchmarks on different cores.

The "Workload Automation" tool [6] was used to evaluate the possibility of automating common daily scenarios. The phone is rooted, and changes are made to the Linux kernel to facilitate the measurements. We utilize the methodology illustrated in Figure 1, which is similar to experimental methodology in prior work [12].

The steps for obtaining performance and power measurements consist of the following:

(i) Start Power measurement on PC1

(ii) Start local data collection script on the phone through Android Debug Bridge (ADB) terminal on PC2

(iii) Interact with the phone according to the scenario

(iv) Kill phone script

(v) End power measurement on PC1.

The workload automation tool runs on PC2. The instrumentation for the measurements was done with the help of the phone vendor. Non-disclosure agreements with the vendor prevent us from mentioning specific information on the phones or specific raw power numbers. Some of the studies such as measurement of IPC and speculative instruction count were based on hardware performance counters. The Sysbench study was done on an Odroid development board [5], whereas all other studies were done on actual phones.

The big.LITTLE architecture requires one little core to be online to take care of operating system and background tasks. Therefore, one or more cores of interest plus one little core are left on while all other are turned off. In some of the experiments, dynamic voltage and frequency scaling (DVFS) is on and we measure utilizations and frequency residencies. In other experiments such as performance counter based experiments, DVFS is disabled and the cores are set to 1.2 GHz to create a more controlled and direct comparison. Additionally all power gating and clock gating states are disabled in experiments with performance counters to prevent counters from not being read. Tasks are affinitized to the core of interest and the single core version of Geekbench 3 workloads are used.

4 OBSERVATION

In this section, we present answers to various questions that were previously outlined based on the explorations we conducted on the smart phones.

4.1 Big versus Little Cores

In an experiment using performance counters on a smart phone (Platform 1), we explored the IPC differential between the big and little cores for the programs in the Geekbench 3 suite. Figure 2 presents the IPCs. SGEMM and DGEMM provide approximately 3× performance on the big core while AES, Stream Add, and Stream Triad achieve more than 2× speedup. Most of the Geekbench programs show at least 75% performance improvement while running on the big core instead of the little one. The lowest improvement is seen in Blur Filter, a 20% improvement, which is still fairly significant in the CPU design world. The big core supports out-of-order execution and seems to get significant improvement in many applications.

Key Insight: The big core can provide 1.2× to 3× performance compared to the little core. This performance advantage may be important for delivering required performance in latency sensitive applications.

4.2 Core Count and Utilization

Different companies have their own answers for how many cores are required in a smart phone: 2, 4, 8, or even 10 cores. With more knowledge of actual user behavior, the optimal core configuration can be explored. This includes the number of cores and the type of cores.

The majority of the scenarios investigated did not significantly stress the big cores. Games and virtual reality applications need compute capability, but are mostly mapped to the GPU. Most common everyday scenarios produced a profile similar to Figure 3. The big cores are typically used in the initial phases (mainly applaunch - not shown in these figures due to the coarse sampling period) and then switched to very low activity mode. The little cores have periods of low activity and high activity, but rarely does the CPU reach full utilization (400% considering the 4 cores).

Figure 4 shows core frequencies and utilizations over time under for a multiplayer game application that was investigated (Clash Royal). For each core type, utilization is plotted as the raw sum or the sum weighted by core frequency over all 4 cores, averaged over each sample period. No heavy utilization of the phone CPUs is seen. Frequency is low throughout the whole duration of game play with most of the computations happening in the cloud (or in the GPU), not on the mobile platform itself.

The DVFS governors switch the cores between the various frequencies depending on the utilization. We studied the frequency residencies of the little and big cores in several applications. The typical profile resembles Figure 5. The lowest frequency is labeled $F0$, and the next higher operating points are labeled $F1$, $F2$, $F3$, etc. It may be observed that the big core is in the lowest operating frequency almost half of the time. It is in the highest operating frequency for only about 10-20% of the time. The little core is utilized more than the big core, however, the utilization is far from heavy. The highest frequency is used a quarter of the time, but 1 GHz or more is used more than 50% of the time.

However, as will be shown in more detail later (Figure 6, Figure 8 and Figures 9 through 11), several activities put a high load on the cores for just a small amount of time. These burst workload require all 4 of the big cores running at max frequency but quickly put the big cores back to sleep after the initial burst. After experimenting with the more than 3 dozen applications and scenarios, the applications that demand high utilization and high frequency are identified to be the following:

(1) Application launches: in the applaunch phase, the big cores do go close to the 400% utilization.
(2) Application updates: The particular update scenario that we experimented with updates 6 applications with antivirus enabled (see Figure 8 and Figures 9 through 11). The 6 applications were Chrome, Gmail, Google Drive, Google Play Music, Maps, and Youtube.
(3) Image processing: Rotating an image, burst shot, lapse it application

Key Insight: Individual applications rarely utilize all 4 big cores for large durations of execution, but all 4 big cores are utilized during scenarios such as application launch and also during update of applications. Utilization of big cores and power consumption are high during such phases to provide a satisfactory user experience.

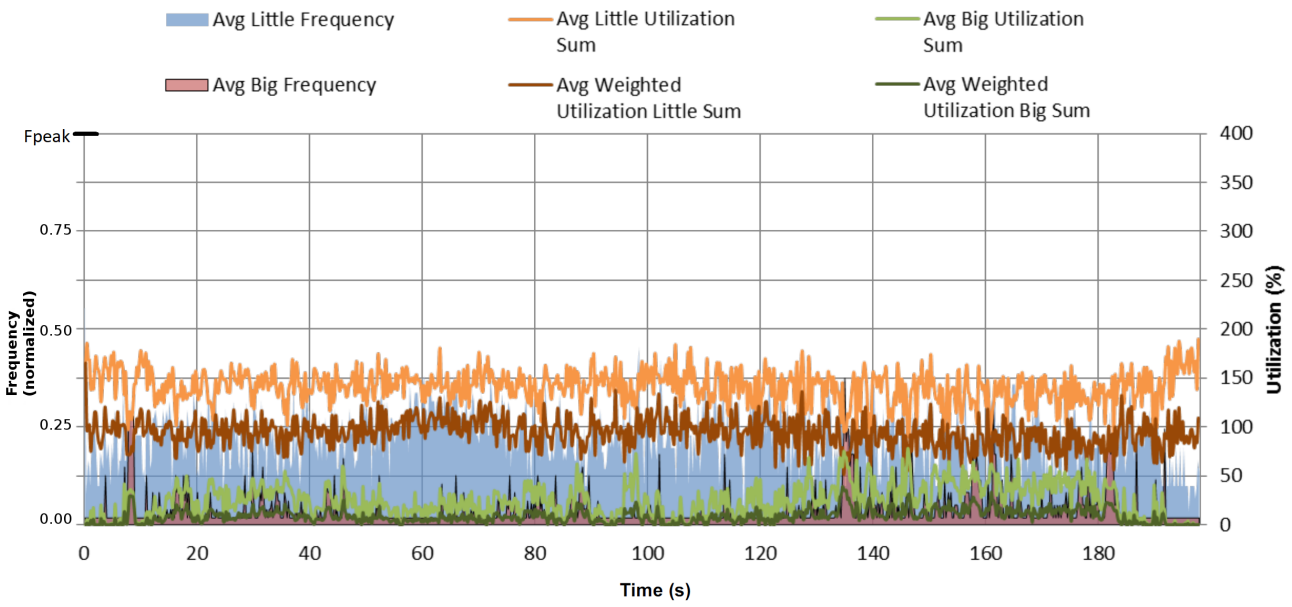


Figure 4: Utilization during Clash Royale multiplayer game, with DVFS on. 400% indicates full utilization on four cores.

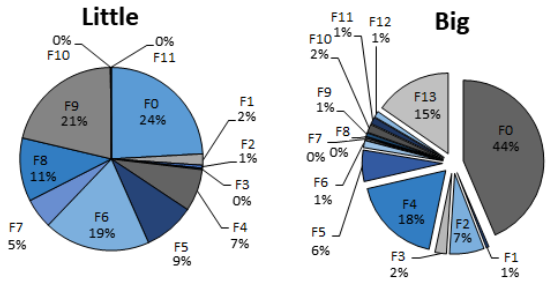


Figure 5: Frequency residencies of big and little cores in example workload (web browsing in 4 Big, 4 Little configuration). F0 is the lowest frequency.

The high activity in application launches and updates can also be seen in any sort of transition. Transitions include any switch from one activity to another: next to application launches, for example, also in loading webpages, returning to the homepage, switching applications, etc. Transitions cause spikes in power due to the quick change to max frequency with a high load as shown in Figure 6, which plots power in the game CrossyRoads. During game play the player is shown various advertisements to earn playing currencies. Anytime the activity switches from gameplay to the advertisement, there is increased power. Each advertisement is likely bringing in a new context and creating increased cache and memory accesses.

Key Insight: App lauches, app updates, image manipulations and advertisements that show up during applications result in high frequencies, high utilizations and increased power usage.

4.3 Ideal Operating Frequencies

The ideal operating frequency for the big and little processors depends on the objective metric. Of course, the lowest frequency is ideal for achieving the lowest power. However, if energy is the objective, low frequencies tend to result in long execution times, which in turn will make the energy consumption higher. We conducted an experiment with SysBench CPU benchmarks to determine the operating frequencies if energy, energy-delay or energy-delay^2 (ED^2) is chosen as the objective.

Figure 7a shows the relative benefit of using the little core versus the big core, or vice versa, from an energy perspective for the SysBench [16] workloads. The energy ratio indicates ratio of big core energy to the energy consumed by the little core. If both cores are operating at 1.4GHz, the big core takes 1.88× the energy of the little core. If the little core is operating at a frequency of 500 MHz and the big core is operating at 2GHz, the big core takes 7.4× the energy than the little one. Similarly, if the big core is operating at 800MHz and the little core is operating at 1400 MHz, the big core consumes 1.45× the energy of the little core.

As you scan the little core (A7) frequencies from left to right, you can see the values increase and then decrease. The highest value (highlighted in green) indicates where the little core is the most energy efficient in comparison to the big core. As you scan the big core (A15) frequencies from left to right, the values decrease and then increase. The lowest value (marked in red) indicates the frequency at which the big core is most efficient. The lines highlighted in green and red are the optimal frequencies for each core. However, one should note that all the numbers in the figure are higher than one, indicating that *the big core is never more energy-efficient than the little one, if low energy is the objective.* In order to get energy benefits, the big core has to be redesigned to be more

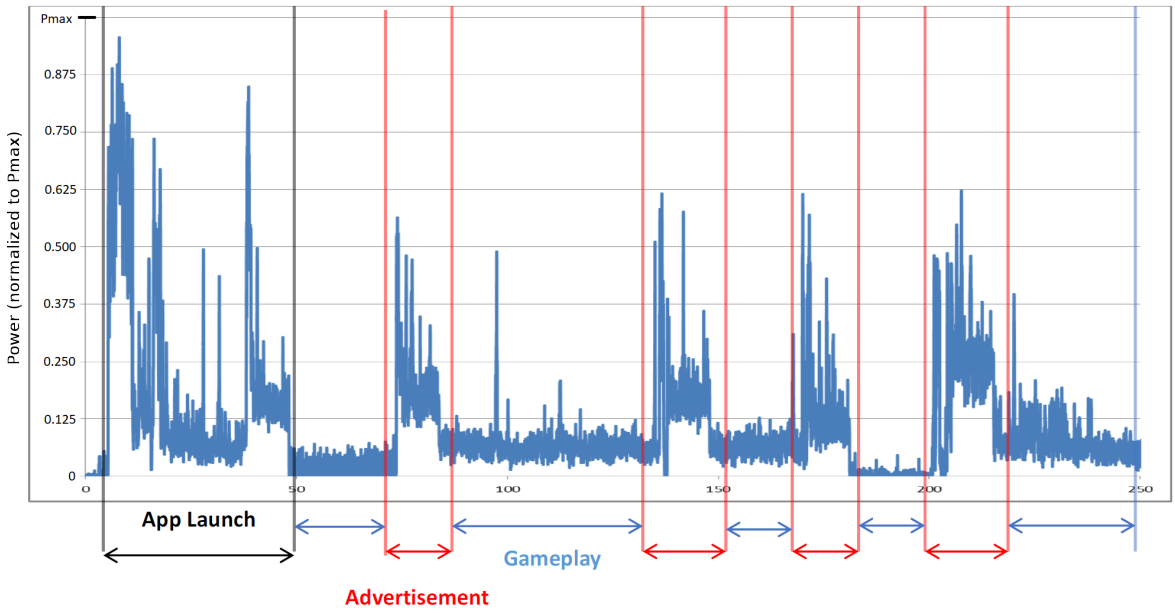


Figure 6: Power consumption over the course of Crossy Road gameplay. Power is high during applaunch and periodic advertisements (this experiment was done on Platform 2).

power-efficient. A design challenge would be to design a big core that can give ratios in this table that are smaller than 1. The core can spend higher power, but its performance has to be so much higher that the overall energy is less.

In some scenarios, quality of service (QoS) demands may be higher and energy may not be the objective metric. Energy Delay Product (ED) may be the metric of interest. Figures 7b and 7c illustrate the data from the EDP and ED^{2}P perspectives. If energy delay product is the objective metric, the optimal frequency of operation of the little core is 1.3GHz and the optimal frequency of operation of the big core is 1.5 GHz. If (ED^2) is the metric of interest, the little core should operate at 1.4 GHz (its highest frequency) and the big core should operate at 1.8GHz. The interesting observation from Figure 7c is that there are several numbers in this table with values less than 1, i.e. if energy-delay^2 (ED^2) is the metric of interest, the big core is more efficient than the little core at several operating frequencies.

Key Insight: If energy is the objective metric, ideal operating frequencies are very low (eg: 500 MHz and 800 MHz are the most efficient for the little and big core respectively). The big core is never more energy efficient than the little core if energy is the metric of interest. However, if (ED) or (ED^2) are the metrics of interest, the big core is more efficient than the little core at various operating frequencies. The ideal operating frequency for the big core increases from 800 MHz to 1.5 GHz and 1.8 GHz if (ED) or (ED^2) are the metrics of interest. In the phases the big cores are not required, they should be gated to be off.

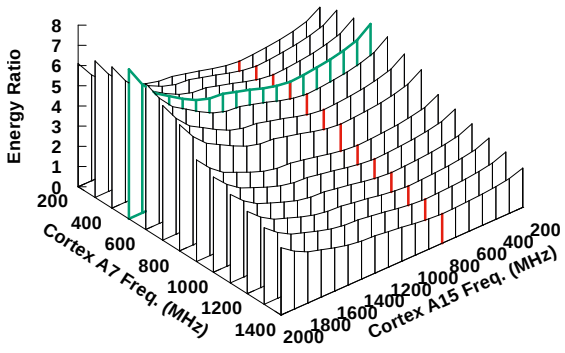
4.4 Race-to-Idle using More Cores for Energy Efficiency

As obviously understood, fewer cores will save power compared to more cores, but the execution time is likely to go up, at least in compute-intensive applications. The increased execution time will result in energy being consumed for longer time, leading to interesting tradeoffs in energy versus performance. In prior sections, we showed that a larger number of big cores can be beneficial for performance. To explore whether computing with fewer number of cores is efficient from the energy-perspective, update and reboot scenarios were studied in the reduced core context.

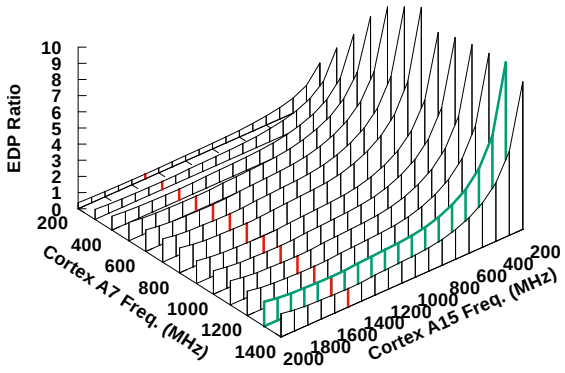
The update scenario updates Chrome, Gmail, Google Drive Google Play Music, Maps, and You tube with antivirus enabled. The update scenario was chosen because it is computationally heavy. The number of cores was varied from 1 to 2 to 4 cores for the update scenario. Figure 8 presents the power and energy of the 2 core and 1 core case in comparison to the 4 core scenario. In the 2 core versus 4 core experiment, it was seen that 4 cores take more power but less energy, with the 2 cores consuming 30% more time and 17% more energy. In the experiment with 1 core versus 4 cores, 1 core takes 118% more time and 46% more energy than if 4 cores were employed. Hence we conclude that it is beneficial to have large amounts of computing capability on the phone since there are scenarios with larger computing needs, where the increased computing ability not just results in better performance but also in energy-efficient operation.

Key Insight: Application updates with more cores yields performance and energy savings although power is higher.

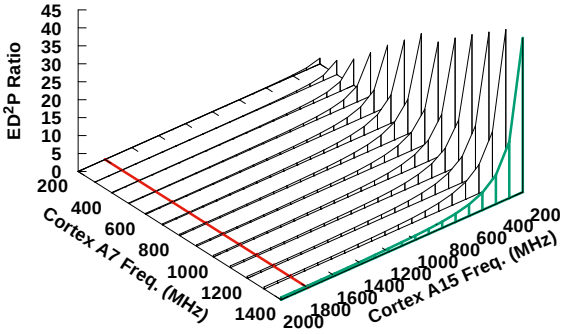
Compared to other applications, the utilization is seen to be heavy in the update scenario. In the 4-core case, all 4 cores are



(a) Energy comparison between Cortex A15 and Cortex A7



(b) EDP comparison between Cortex A15 and Cortex A7



(c) ED^2P comparison between Cortex A15 and Cortex A7

Figure 7: Relative energy efficiency of big versus little core.

utilized for extended periods of time. The scenario gets thermally limited as illustrated in Figure 9 and Figure 10. Thermal throttling is triggered when the temperature reaches a preset control temperature. One can observe that thermal throttling happens in both 4-core and 2-core runs of the update scenario. Once temperature reaches the control temperature, frequency goes down due to thermal throttling, and utilization goes up. Throttling leads to lower temperatures and frequency increases again. This phenomenon periodically repeats. In the 1-core run illustrated in Figure 11, temperature never exceeds the control temperature. However, the performance is heavily degraded.

While updates may happen in the night when the phone is charging and hence this may not appear to be a serious concern,

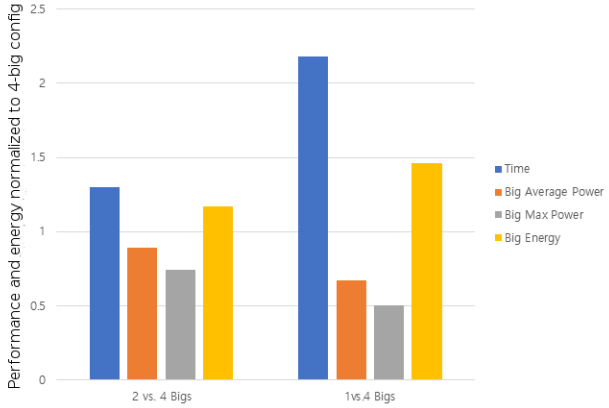


Figure 8: Comparison of 2-core and 1-core configurations to 4-core. More cores are a win for both performance and energy in the update scenario (data from Platform 2).

this finding applies to any workload scenario with heavy utilization. While many of the workloads studied earlier showed low utilization, under multitasking and future applications, it is possible that such a scenario will exist under normal operation.

Another common question is whether significant savings in power can be obtained if slow reboots can be tolerated. Reboot was done utilizing all 4 cores, with 2 cores and with only 1 core (Table 3). The booting average power reduces more than 50% if the user is willing to tolerate a boot using only 2 big cores taking 38% more time. The booting average power further reduces to 37% of the original power if the user is willing to tolerate a boot that takes 2.4× the original boot time. The energy consumed for boot in the 2-core case is only 62% of the default boot energy if the 38% additional boot time can be tolerated. The boot with the single core consumes 90% of the default boot energy making the 2-core boot the most energy-efficient.

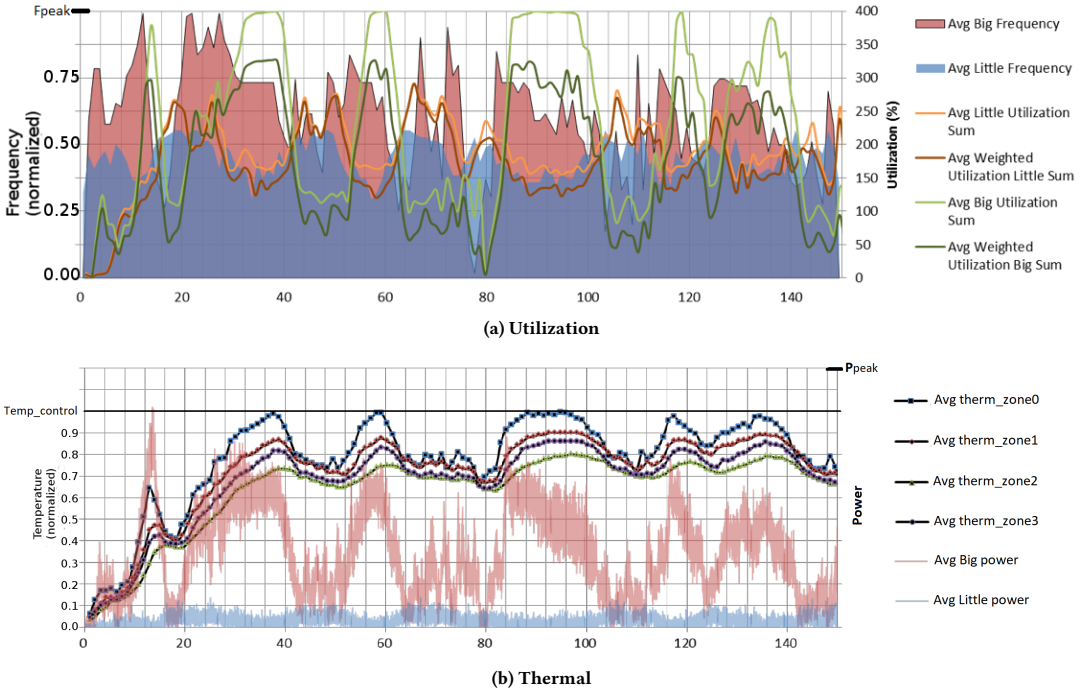
Key Insight: Rebooting with more cores yields the best performance, however energy efficiency is better with fewer cores. Note that the performance/energy tradeoffs in rebooting are different from the update scenario, where more cores are better for performance *and* energy.

4.5 Importance of Branch Prediction for Mobile Workloads

Another interesting question is whether mobile applications need a sophisticated branch predictor. Many mobile apps include loops that do repeated processing of streams of data. Considering that loop branches are easy to predict, is sophisticated branch prediction necessary for a mobile processor? Investigating Geekbench benchmarks using hardware performance counters, we see that several

Table 3: Reboot power/latency tradeoffs (Platform 2).

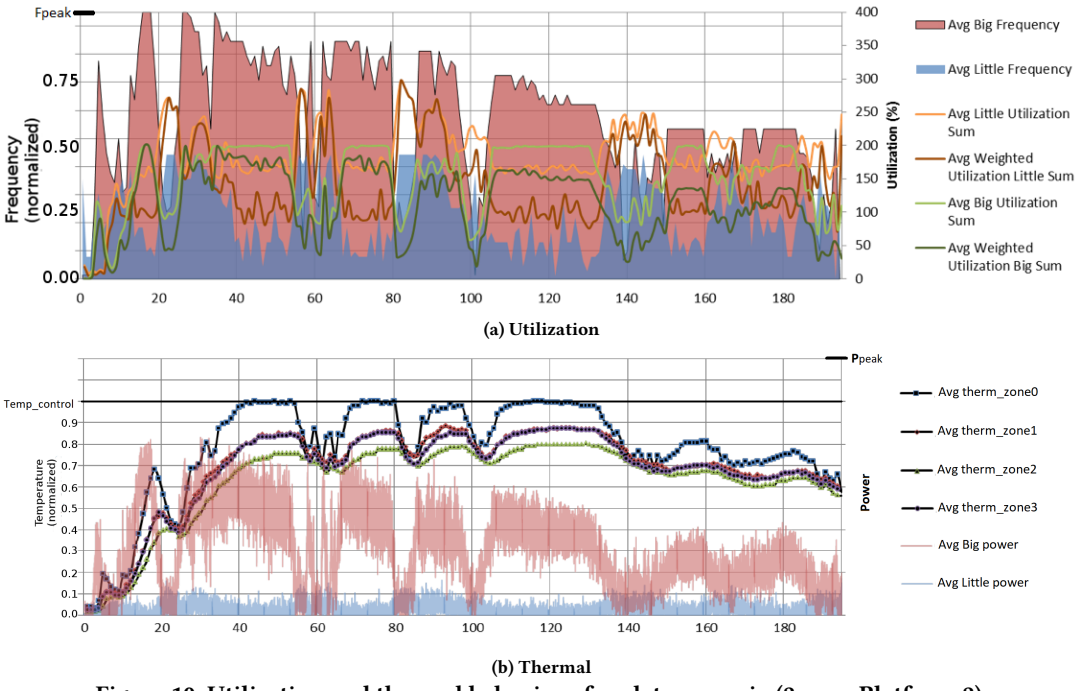
Boot Scenario	Average Power (W)	Boot Time (s)
4 core	P	T
2 core	0.45P	1.38T
1 core	0.37P	2.46T



(a) Utilization

(b) Thermal

Figure 9: Utilization and thermal behavior of update scenario (4 core, Platform 2)



(a) Utilization

(b) Thermal

Figure 10: Utilization and thermal behavior of update scenario (2 core, Platform 2).

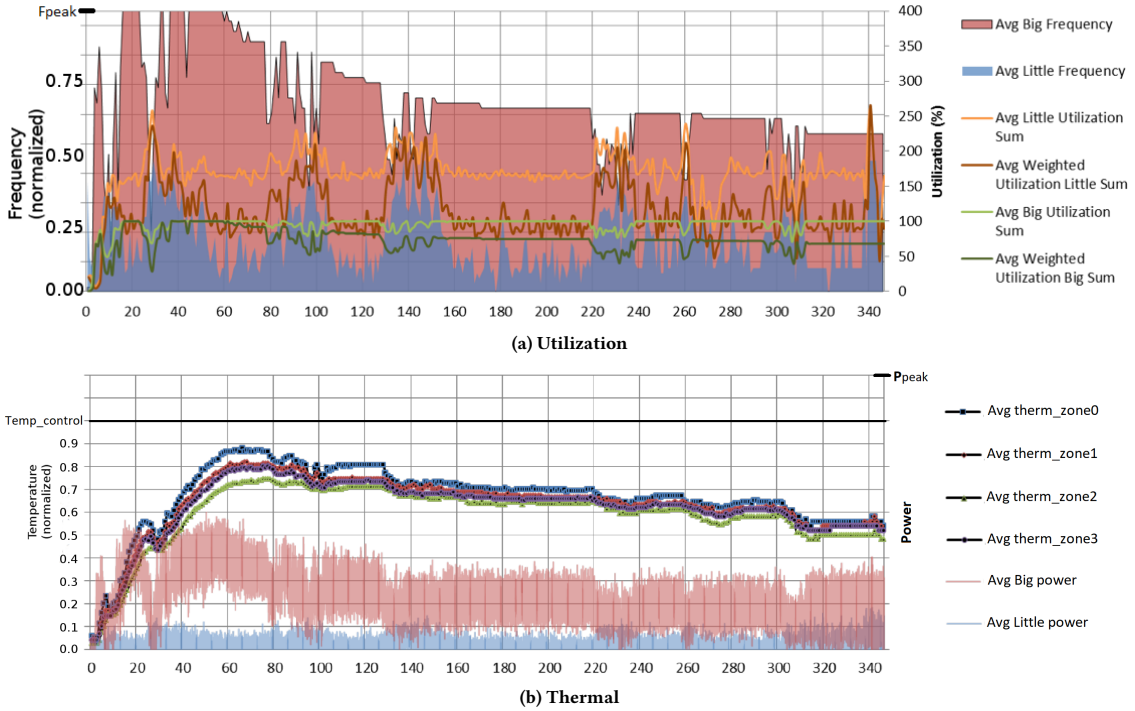


Figure 11: Utilization and thermal behavior of update scenario (1 core, Platform 2).

workloads result in a large percentage of branch mispredictions. The impact of the mispredicted branches can be understood if one analyzes the number of wastefully fetched/decoded/processed instructions. Figure 12 shows the number of speculatively executed instructions over the instructions retired in Platform 1. This graph highlights several workloads that are doing excessive work which get discarded later, indicating significant mis-speculation.

BZip compress, PNG compress, Lua, Dijkstra, and N-Body all execute at least an extra 34% instructions. BZip2 Decompress, JPEG Compress and PNG Decompress execute at least 20% extra instructions. The extra work in fetching, decoding and executing these extra instructions translates into wasted energy and battery life.

Going further, Figure 13 shows the correlation between branch misprediction and the speculative instruction ratio (for Platform 1). As expected, there is a strong correlation, but one surprising trend is the amount of instructions generated by such a small amount of mispredictions. Following the trend line, 10 mispredictions per thousand instructions generates roughly 25% more speculative instructions. Increasing the mispredicts by 8 per thousand instructions leads to almost 50% more instructions. This seems to indicate that there is a large penalty for branch mispredictions with many instructions flushed.

Key Insight: Speculative instruction and energy overhead of mis-predicted branches can be severe for mobile workloads despite them being perceived as loop-intensive. This suggests that some mobile applications can benefit from a sophisticated branch predictor and/or mechanisms to reduce the penalty of branch mis-predictions.

4.6 IPC as Performance Metric

Geekbench scores are popular for expressing smart phone performance. While working on processor characteristics, especially in early design tradeoff evaluation, architects often use IPC as an indication of performance. But what is the effect of IPC on Geekbench scores? Does Geekbench score increase if IPC is improved?

Utilizing performance counters on benchmarks provides an opportunity to search for correlation between hardware events and benchmark score, i.e. to evaluate basic micro-architecture measurements, such as IPC, as general performance metrics. As shown in Figure 14, there is a very high correlation between the percent difference in the Geekbench 3 score and the percent difference in the IPC of each benchmark between a big and little core. Each point is a different workload in the figure. This trend is encouraging, as it indicates that IPC may be used to derive insights during processor design. Originally, the raw IPC delta was used, which yielded a very poor correlation, but once the percent difference in IPC with respect to the base IPC was used, a good correlation was observed.

Figure 2 earlier showed the change of IPC in each Geekbench 3 workload between a big and a little core. Since IPC is correlated to the Geekbench 3 score this graph shows that focusing on microarchitectural changes to a workload such as Blur Filter will yield little improvement.

Key Insight: While absolute IPC changes are a poor predictor of performance effects on benchmark scores, relative IPC can serve as a performance metric for studying micro-architecture impact on mobile applications.

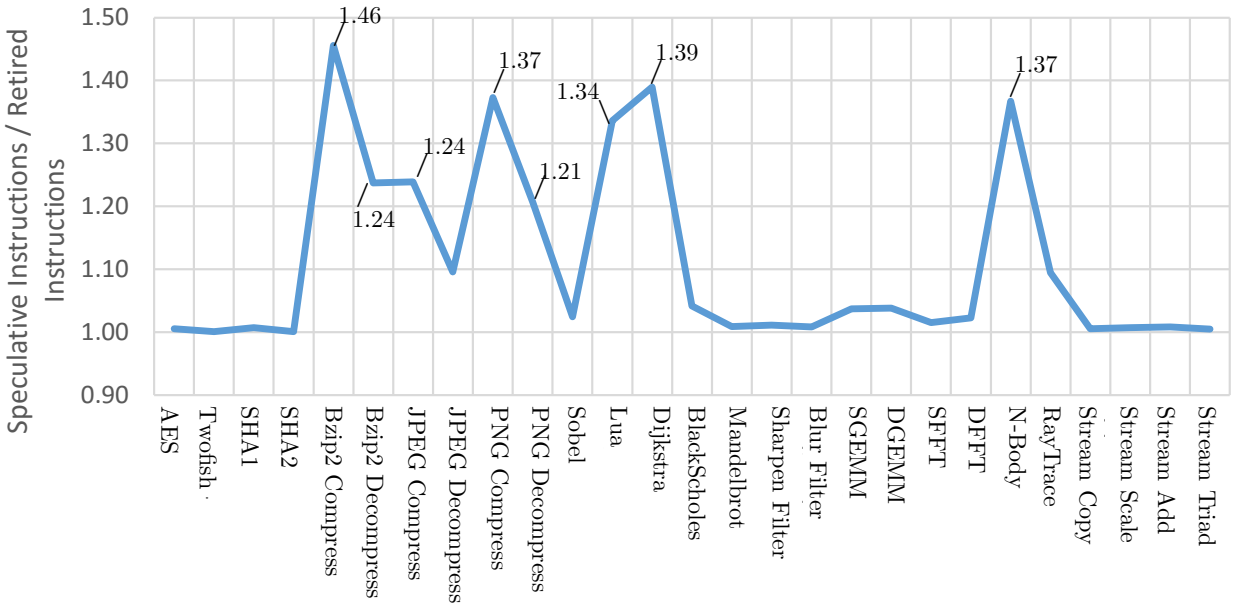


Figure 12: Speculative instruction ratio across Geekbench 3 workloads (Platform 1).

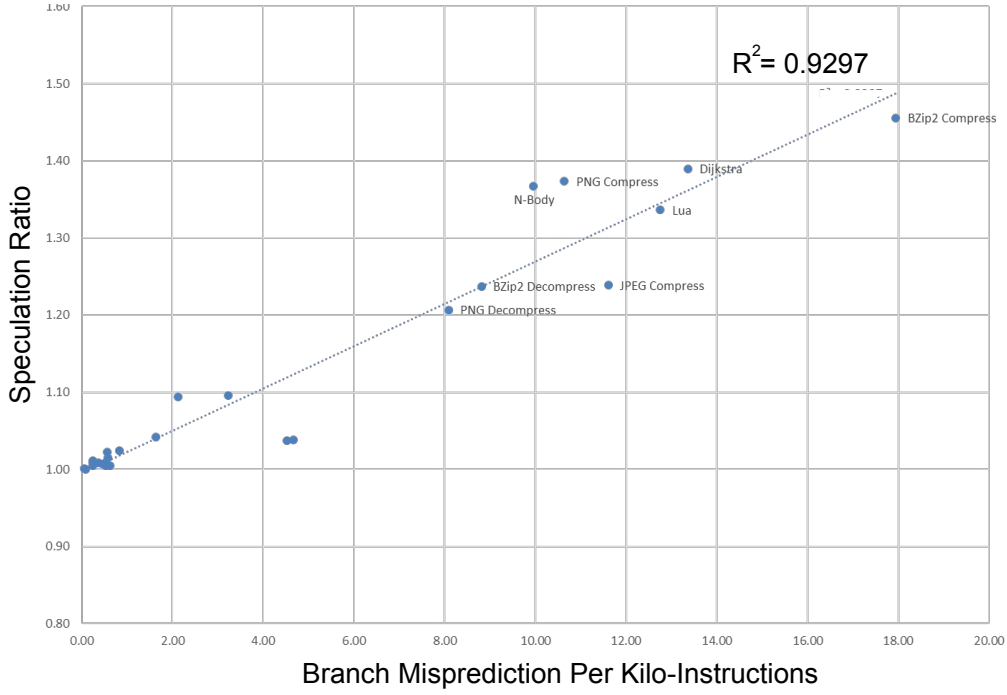


Figure 13: Speculative instruction ratio vs. branch mispredictions per kilo-instruction across Geekbench 3 workloads.

5 CONCLUSION

In this paper, we presented execution characteristics of current generation smart phone platforms in order to draw insights for next generation smart phone CPU designs. Using more than 3 dozen popular Android applications and two benchmark suites, we examine the performance and power/energy characteristics of multicore smart phone processors with 4 big and 4 little cores. Very few applications can utilize all the 4 big cores simultaneously, however, there are application phases in which all 4 big cores are essential to providing the required performance. We also find that

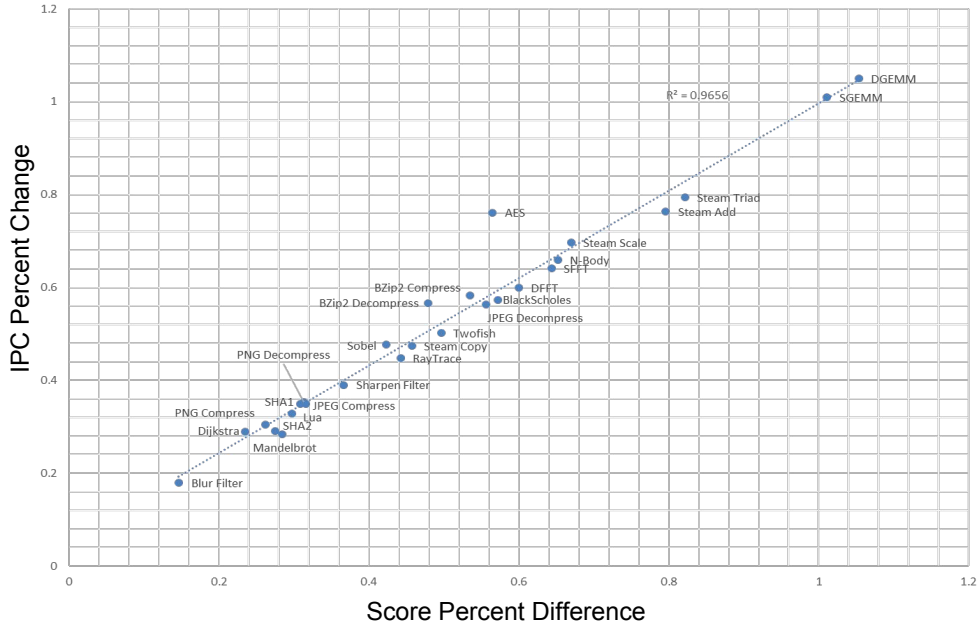


Figure 14: Percent difference of IPC between the big and little cores vs. percent difference of Geekbench 3 score between the big and little cores.

at least in some applications, it is more energy-efficient to use all cores and finish the task quickly. Application launching and update operations are two scenarios that can obtain higher energy efficiency and performance if all 4 cores are used. Aggressively using all 4 cores can lead to high power consumption and even thermal throttling. Transitioning between applications and even advertisements during applications also introduce bursts with an increase in power consumption.

In light of these observations, we argue that the big cores are very useful for a smart phone platform, but what is essential is an operating system/run time that can decide to use them selectively in the required phases, while reducing their frequencies or turning them off when not needed. Furthermore, architectures should have support for bursty operations and transitions if those are going to be important in the smart phone domain. The insights from this study are expected to be useful in designing next generation smart phone systems.

REFERENCES

[1] 2019. Determining the TDP of Exynos 5 Dual. http://www.anandtech.com/show/6536/arm-vs-x86-the-real-showdown/13.

[2] 2019. Exynos. http://www.anandtech.com/show/9330/exynos-7420-deep-dive/2.

[3] 2019. GeekBench Suite. https://geekbench.com.

[4] 2019. In-depth with the Snapdragon 810's heat problems. https://arstechnica.com/gadgets/2015/04/in-depth-with-the-snapdragon-810s-heat-problems/.

[5] 2019. Odroid-XU3 Board. http://www.hardkernel.com.

[6] 2019. Workload Automation Tool Suite. https://github.com/ARM-software/workload-automation.

[7] Rizwana Begum, David Werner, Mark Hempstead, Guru Prasad, and Geoffrey Challen. 2015. Energy-Performance Trade-offs on Energy-Constrained Devices with Multi-component DVFS. In *IEEE Int. Symp. on Workload Characterization (IISWC)*.

[8] Carole-Jean Wu Benjamin Gaudette and Sarma Vrudhula. 2016. Improving Smartphone User Experience by Balancing Performance and Energy with Probabilistic QoS Guarantee. In *International Symposium on High Performance Computer Architecture (HPCA)*.

[9] Dileep Bhandarkar and Jason Ding. 1997. Performance characterization of the Pentium Pro processor. In *Intl. Symp. on High-Performance Computer Architecture (HPCA)*.

[10] Guilin Chen, Ozcan Ozturk, Guangyu Chen, and Mahmut Kandemir. 2006. Energy-aware code replication for improving reliability in embedded chip multiprocessors. In *International SOC Conference*.

[11] G Chen, Liping Xue, Jungsub Kim, Kanwaldeep Sobti, Lanping Deng, Xiaobai Sun, Nikos Pitsianis, Chaitali Chakrabarti, M Kandemir, and Narayanan Vijaykrishnan. 2006. Geometric tiling for reducing power consumption in structured matrix operations. In *International SOC Conference*.

[12] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. 2017. Determining Application-specific Peak Power and Energy Requirements for Ultra-low Power Processors. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[13] Cao Gao, Anthony Gutierrez, Madhav Rajan, Ronald G. Dreslinski, Trevor Mudge, and Carole-Jean Wu. 2015. A Study of Mobile Device Utilization. In *Intl. Symp. Performance Analysis of Systems and Software (ISPASS)*.

[14] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. 2013. RERAN: Timing- and Touch-sensitive Record and Replay for Android. In *Proc. of International Conference on Software Engineering (ICSE)*.

[15] A. Gutierrez, R.G. Dreslinski, T.F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver. 2011. Full-System Analysis and Characterization of Interactive Smartphone Applications. In *IEEE Int. Symp. on Workload Characterization (IISWC)*.

[16] Alexey Kopytov. 2019. Sysbench CPU Benchmark Suite. https://launchpad.net/sysbench.

[17] Dhinakaran Pandiyan, Shin-Ying Lee, and Carole-Jean Wu. 2013. Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite - MobileBench.. In *IEEE Int. Symp. on Workload Characterization (IISWC)*.

[18] Partha Ranganathan. 2017. ISCA 2017 Keynote Speech, Video shown during the keynote speech, Toronto, Canada, June 25 2017. *ACM ISCA* (2017).

[19] Wonik Seo, Daegil Im, Jeongim Choi, and Jaehyuk Huh. 2015. Big or Little: A Study of Mobile Interactive Applications on an Asymmetric Multi-core Platform. In *IEEE Intl. Symp. on Workload Characterization (IISWC)*.

[20] Karthik Swaminathan, Emre Kultursay, Vinay Saripalli, Vijaykrishnan Narayanan, Mahmut Kandemir, and Suman Datta. 2011. Improving energy efficiency of multi-threaded applications using heterogeneous CMOS-TFET multicores. In *International Symposium on Low Power Electronics and Design (ISLPED)*.

[21] Yuhao Zhu, Matthew Halpern, and Vijay Janapa Reddi. 2015. The Role of the CPU in Energy-Efficient Mobile Web Browsing. *IEEE Micro* 35, 1 (2015), 26–33.