Data Movement Energy Characterization of Emerging Smartphone Workloads for Mobile Platforms

by

Dhinakaran Pandiyan

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved November 2014 by the
Graduate Supervisory Committee:

Carole-Jean Wu, Chair
Yann-Hang Lee
Aviral Shrivastava

ARIZONA STATE UNIVERSITY

December 2014

ABSTRACT

A benchmark suite that is representative of the programs a processor typically executes is necessary to understand a processor's performance or energy consumption characteristics. The first contribution of this work addresses this need for mobile platforms with MobileBench, a selection of representative smartphone applications. In smartphones, like any other portable computing systems, energy is a limited resource. Based on the energy characterization of a commercial widely-used smartphone, application cores are found to consume a significant part of the total energy consumption of the device. With this insight, the subsequent part of this thesis focuses on the portion of energy that is spent to move data from the memory system to the application core's internal registers. The primary motivation for this work comes from the relatively higher power consumption associated with a data movement instruction compared to that of an arithmetic instruction. The data movement energy cost is worsened esp. in a System on Chip (SoC) because the amount of data received and exchanged in a SoC based smartphone increases at an explosive rate. A detailed investigation is performed to quantify the impact of data movement on the overall energy consumption of a smartphone device. To aid this study, microbenchmarks that generate desired data movement patterns between different levels of the memory hierarchy are designed. Energy costs of data movement are then computed by measuring the instantaneous power consumption of the device when the micro benchmarks are executed. This work makes an extensive use of hardware performance counters to validate the memory access behavior of microbenchmarks and to characterize the energy consumed in moving data. Finally, the calculated energy costs of data movement are used to characterize the portion of energy that MobileBench applications spend in moving data. The results of this study show that a significant 35% of the total device energy is spent in data movement alone. Energy is an increasingly important criteria in the context of designing architectures for future smartphones and this thesis offers insights into data movement energy consumption.

ACKNOWLEDGEMENT

The first person I would like to thank is my advisor, Dr. Carole-Jean Wu. Dr. Wu has been a great source of inspiration with her hard work and passion for computer architecture. She kindled my interest in research and has helped me sustain that interest by offering challenging problems to work on. She has been instrumental for me to learn the significance of paying attention to details. I owe most of my learning during the past two years to the opportunities Dr. Wu has provided me. I am also immensely grateful for her patience to improve my thesis. This thesis would not have been possible without her support and mentoring.

I am grateful to my committee members Dr. Aviral Shrivastava and Dr. Yann-Hang Lee for their valuable suggestions that have made my thesis better.

I want to thank my friends Akhil, Jeevan, Saketh and Shin-Ying for helping me stay motivated through tougher times.

Most importantly, I am thankful to my family for their vital and unrelenting support.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

CHAPTER

1 INTRODUCTION 1

1.1 Background and Motivation 1

1.2 Thesis Contributions 4

1.2.1 Outline 6

2 RELATED WORK 7

3 MOBILEBENCH 11

3.1 Workloads 11

3.2 Platform Energy Characterization with EnergyUsageCollector 13

4 MICROBENCHMARKS 18

4.1 Background 18

4.2 Design of the Microbenchmarks 18

4.3 Discussion 24

5 DATA MOVEMENT ENERGY COST EVALUATION 26

5.1 Dynamic Energy Measurement for the Microbenchmarks 26

5.2 Stall Cycles 28

5.3 Long Latency Memory Operations 29

5.4 Cache Prefetcher 29

6 EXPERIMENTAL SETUP 31

6.1 Experimental Platform 31

6.2 Energy Measurement for the Experimental Platform 31

6.3 Performance Counters 33

CHAPTER Page

7 DATA MOVEMENT CHARACTERIZATION FOR MOBILE WORKLOADS 35

8 CONCLUSION 40

8.1 Summary of Results 40

8.2 Future Research Directions 40

8.2.1 Smartphone processors 40

8.2.2 Data movement in heterogeneous mobile systems 41

8.2.3 Data movement in client-server environments 43

REFERENCES 44

LIST OF TABLES

Table Page

6.1 Technical Specifications of the System. 32

7.1 Energy Cost of Data Movement. 36

LIST OF FIGURES

Figure Page

1.1 Power Consumption Comparison for Load and Add 5
3.1 Power Profile for Hardware Components in Galaxy S3 14
3.2 MobileBench Energy Profile 15
4.1 Galaxy S3 Architecture 19
4.2 Memory Access Latency 20
5.1 Energy Measurement for Data Movement 27
6.1 Current and Voltage Measurement with NI SignalExpress 33
6.2 ARM Streamline Timeline View 34
7.1 Energy Breakdown for the Experimental Device 36
7.2 Relative Energy Costs of Data Movement 38

Chapter 1

INTRODUCTION

1.1 Background and Motivation

The smartphone market has witnessed a rapid growth in the past decade. The sales figures for smartphones [41] surpassed that of desktops [42] in 2010 and touched one billion units in annual sales in 2013 [40]. Touch-sensitive high resolution displays, high speed internet connectivity, powerful processors, intuitive operating systems and large repositories of applications have all aided the proliferation of these devices. Smartphones have now become sufficiently capable to handle a wide variety of use cases for which desktops were earlier used. For example, the Microsoft Office productivity suite [27] has been adapted to work on smartphones in 2014. Smartphones nowadays are adept at handling word processing, high quality media consumption, high resolution image and video capture, gaming, navigation, web browsing and many other use cases. To support the ever-growing list of applications [43], smartphone manufacturers have been steadily increasing the computing horsepower of these devices. Some of the mechanisms that have been widely adopted in the industry, especially in top-of-the-line devices, are the integration of an increasing number of general-purpose application cores (up to eight cores as of 2014) [30], more than one hundred graphic processing cores [22], and many special purpose accelerators such as digital signal processors and video encoder/decoders. This is exemplified by the recent Exynos Octa [30] or the Snapdragon 810 SoCs [35]. The application cores in these modern smartphones have also grown more complex, with a goal to deliver high performance since users expect feature-rich, responsive and interactive applications to operate seamlessly.

The processors that are used in smartphones have to be vastly different from their desktop counterparts to cater to the interactive applications that they handle and operate at a different power-performance point. Benchmark suites like SPEC CPU2006 [18] and PARSEC [11] are widely used across the industry and academia in order to evaluate the performance and power consumption of processors in the desktop and server space. However, there are no such comparable benchmark suites targeted at mobile platforms. The mobile platform benchmark suite, **MobileBench**, was hence developed with a goal to explore key characteristics critical to interactive applications, to examine the effectiveness of modern architectural features in the hardware, and to design architectural features specifically for smartphone devices [33]. MobileBench is useful for comparative studies of different mobile devices from either the performance or energy stand point. MobileBench includes representative smartphone applications including general-purpose interactive web browsing, education-oriented web browsing, photo browsing and video playback applications, which constitute a majority of activities performed on today's smartphones.

From the energy perspective, smartphones much like other portable electronic devices have to work with a limited energy capacity constrained by the battery. Additionally, smartphones are designed to be light and thin which limits the size of the Li-Ion batteries. For example, many top-of-the-line smartphones today that weigh less than 150 grams, have batteries with a capacity of around 2000mAh. The battery technology for smartphone devices has not kept pace with developments that some of the other components like processors or displays have seen. While intuitively, the display is expected to be the major source of battery drain, it is imperative to fully understand the energy consumption characteristics of the device. To that measure, energy consumption of various important platform components on a smartphone is profiled with an Android background service called EnergyUsageCollector. The relevant details are provided in the Chapter 3. Coarse-grained profiling of MobileBench applications with EnergyUsageCollector demonstrates that when the brightness of the LCD screen is at 25% of its maximum value, the application cores become the most energy-hungry element, consuming more than 50% of the total energy capacity. This suggests that the energy consumption characteristics of the application cores, which consume significant energy, has to be studied further.

Most prior works that characterize the power profile of smartphones focus on component-level results. However, component-level energy characterization does not give the full picture of how energy is spent in the entire system; the information about where data resides and how data is moved across the system have not been considered. A recent work has highlighted the significance of the data movement energy cost in the server computing environment [24] concluding that 28-40% of total processor energy consumption executing scientific applications is spent on data movement. Moving data present in the cache requires as much energy as a floating point computation itself and costs much more if the data is not in the cache hierarchy [23]. The gap between the energy cost of moving data from memory to registers and the energy cost of performing floating point computations is expected to widen for future systems. The energy cost of double-precision floating point operations is expected to reduce by ten times by 2018 while the energy cost of moving data from memory to register is expected to remain the same [2, 14]. This trend highlights the importance of data movement energy.

Interestingly, such analysis has not been performed for smartphone processors specifically. Smartphone SoCs employ heterogeneous units for different types of computations. With many components generating, processing, and consuming data within the SoC, the on-chip data traffic is bound to be high. This data sharing happens to be primarily accomplished via the system memory. In light of the significance of data movement energy consumption for scientific applications in servers, the application core's power consumption and distributed computing models adopted in smartphones, a detailed investigation is carried out to quantify the energy cost of data movement for modern smartphone applications on a real, commercial device. To begin with, it has been found in this thesis that even when the mobile application processor is solely working on fetching data from the memory and spending most of the cycles waiting for data, its power consumption is on par with it busy executing arithmetic operations under 100% utilization. Figure 1.1 compares the application processor power consumption under four different scenarios: continuous load instruction execution with and without data dependency, and add instruction execution utilizing one or both pipelines. This experiment was performed with a dual-issue ARM Cortex-A9 processor that is present in the Galaxy SIII device. The inference drawn from this experiment is that the dynamic power consumption of the benchmark which performs continuous, independent load operations is far more significant than that of the benchmark performing continuous add operations. This illustrates the significance of the data movement energy cost relative to ALU operations and its potential impact on total application energy consumption in mobile platforms.

1.2 Thesis Contributions

Overall, this thesis makes the following contributions:

1. MobileBench, a benchmark suite of representative smartphone applications that enables comparative performance and energy evaluations in modern smartphone SoCs is created. With the availability of the MobileBench suite, a background service application in the Android framework called EnergyUsageCollector is

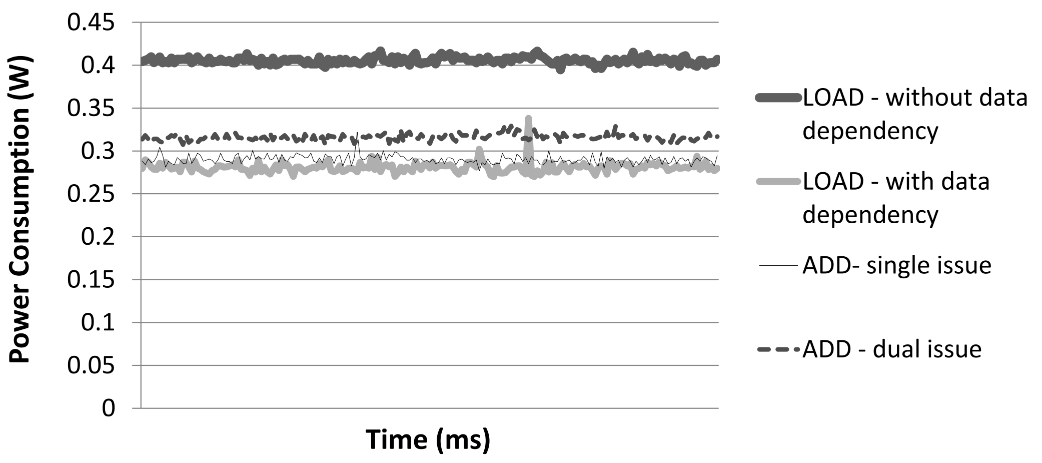


Figure 1.1: Dynamic power consumption comparison for load instruction execution with and without data dependency, and add instruction execution utilizing one or both pipelines of the dual-issue ARM Cortex-A9 processor.

designed to characterize the energy consumption behavior for modern smartphones.

2. A microbenchmark based methodology is proposed to characterize the data movement energy across the different levels of the memory hierarchy on a commercial real-device. Hardware performance counter statistics and power meter measurements are utilized to quantify the energy cost of data movement.

3. A detailed characterization is performed to show the significance of data movement energy and stalled cycle energy when running realistic smartphone applications, including various web browsing activities, video playback, photo browsing, and an interactive game. Based on the results, this work offers insights for future smartphone architecture designs.

In summary, the results presented in this thesis show the energy cost to perform a memory load instruction whose data is not found in any level of the cache hierarchy is **115** times higher than that of an add operation. On average, a significant portion

(34.6%) of the total device energy consumption is spent on moving data from one level of the memory hierarchy to the next level for the interactive smartphone workloads. The data movement energy is particularly high, 41% for realistic web browsing [32].

1.2.1 Outline

The remainder of this thesis is organized as follows: Chapter 2 discusses related work. Chapter 3 describes MobileBench and component energy profiling using EnergyUsageCollector. In the Chapter 4, the background information of the memory hierarchy for modern smartphone architectures is provided along with the design of the microbenchmarks that characterize the data movement energy for a specific memory hierarchy. Chapter 5 explains the energy measurement methodology in detail and Chapter 6 describes the real-device experimental setup for measuring the energy consumption. Chapter 7 concludes the data movement study with data movement energy characterization for MobileBench applications. Chapter 8 summarizes the results and presents future research directions.

Chapter 2

RELATED WORK

There are numerous approaches that researchers take to gain a deeper understanding of the power consumption by processors. One such popular approach is to construct power models that provide a breakdown of the total system power consumption into different components. Further, these power models can be designed at various levels of abstraction - circuit, architecture or system components. Accurate models aid in design space exploration with good estimates of power consumption. Software and hardware can then be optimized for a lower power consumption. Low power optimizations are particularly important and necessary for battery-powered devices. McPAT [26] and Wattch [12] are two widely used frameworks to analyze and optimize microprocessor power dissipation at the architecture level. These frameworks estimate the power dissipated by different functional units within the processor based on an input configuration and detailed statistics from a performance simulator. To obtain power estimates that are meaningful from such frameworks, it is key to set up the framework with a configuration that closely resembles the processor under study. Moreover, it is especially hard to model a real interactive system like a modern smartphone to evaluate the impact of data movement.

Instruction level power models that estimate the power consumed during the execution of a software are also extremely useful to optimize software. Such models are typically constructed by measuring dynamic power consumption of different instructions. The information from the measurements is used to profile programs either offline during development or at run-time. For example, in [39], Sinha, Ickes and Chandrakasan proposed a run-time software energy estimation technique for an

ARM architecture that isolates switching and leakage energy components of the software. They observed that the energy consumption of various instruction classes show very little variation for the StrongARM SA-110 and Hitachi SH-4 embedded processors. But, through characterization that is later presented in this thesis, contrasting results are observed for a modern smartphone platform running a commercial operating system. Hao et al. in [17] suggested an energy estimation technique that can aid developers to write energy-efficient code. The proposed tool combines program-level analysis with an instruction level power model to arrive at an energy estimate. However, such byte-code level energy analysis abstracts out vital information that could be used for potential hardware optimizations and also different memory access instructions can get clubbed together. As it is shown in the Chapter 5 in this thesis, energy cost of loading data from the processor's cache is vastly different from loading data from the DRAM. Lee et al. [25] used a combination of test programs and device power measurements to generate an instruction level power model by employing linear regression. The model was targeted at an ARM7TDMI processor that is used in embedded systems but the analysis left out energy estimation for pipeline stalls and load/store instructions. This is crucial as demonstrated in the Chapter 1.

Further, the above mentioned works do not specifically target modern smartphone platforms. With smartphones emerging as ubiquitous computing devices and the architecture of smartphone processors showing trends of increasingly differentiating itself from that of conventional PC processors, it is interesting to study their power and energy consumptions. Several recent studies have been performed to that end. For instance, Zhang et al. [44] described an online, utilization based estimation tool called PowerTutor that uses a pre-generated component-level power model. The power model is generated by running test programs that stress different components and measuring energy consumption either with an external power measurement instrument or internally by sampling the battery voltage. The tool takes into consideration the power states and utilization of various hardware components like the display, CPU and other components to generate the power estimate. This approach to power dissipation estimation is similar to the battery usage statistics that recent versions of Android implement. The statistics, in this case, are computed by making use of the current draw of different components in the *power_profile.xml* file provided by smartphone vendors. In [38], power consumption characteristics in smartphones are studied from a user perspective. Using the regression-based component level power model, Shye, Scholbrock and Memik showed that the display and CPU are the two largest power consuming components. This is similar to the observation from the EnergyUsageCollector 3.2 proposed in this thesis for MobileBench applications. Additionally, Shye, Scholbrock and Memik performed an end-user sensitivity study of power optimizations by tuning the CPU frequency governor and controlling the display brightness. The suggested optimizations take into account *change blindness* that humans exhibit. In [34], the authors tackled scenarios where utilization based power models, e.g., [38] and [44], are insufficient with a power model that is based on system call tracing. On the other hand, Carroll and Heiser [13] used sensors on a smartphone to measure the power usage of the individual computation components under different workloads. Murmuria et al. [28] demonstrated power usage characterization and developed a power-modeling framework based on the component-level power consumption. However, what these component level power models fail to establish is the dependency of system power consumption on data movement within the system. As presented in Chapter 1, the cost of a load instruction moving data from the closest level of the memory hierarchy to the register far exceeds that of an arithmetic instruction. Addressing this issue, a detailed study quantifying data movement energy costs in scientific applications was presented by Kestor et al.[24].

This work uses CPU performance counters and power measurement techniques to evaluate data movement energy costs in servers. The results showed that scientific applications executed on high-performance desktop/server processors spend 28-40% of total energy consumed in moving data. Distinct from all prior works, the energy characterization results presented in this thesis are for modern smartphones executing interactive emerging applications that are different from scientific applications. This thesis offers insights into the impact of data movement for future smartphone architectures.

Chapter 3

MOBILEBENCH

3.1 Workloads

A key aspect of this work is to create a publicly available benchmark suite that contains a collection of representative interactive smartphone applications to be used by the research community for SoC performance and energy exploration. The MobileBench suite is created to enable such comparative performance and energy evaluations. In addition to the publicly available BBench [16] that is used to represent simple web browsing behavior, four additional commonly-used benchmarks – realistic web browsing, education-oriented web browsing, photo rendering, and video playback - are included in MobileBench. Each of the MobileBench applications is discussed in more detail next.

General Web Browsing (`GWB`): One of the most important smartphone applications is web browsing. In fact, the web browser is one of the most commonly-used interactive applications on smartphones. Many other cross-platform applications are also browser-based. To study the behavior of general-purpose web browsing, Gutierrez et al. [16] constructed BBench which is an offline, automated benchmark to assess the performance of a web browser when rendering a collection of 11 popular websites on the web, including Amazon, BBC, CNN, Craiglist, eBay, ESPN, Google, MSN, Slashdot, Twitter, and YouTube. BBench traverses the collection of the websites repeatedly by loading the web page and scrolling down to the bottom of the web page before proceeding to the next website. In this thesis, BBench is referred to as `GWB` since it is a benchmark which focuses on simple general web browsing behavior.

Realistic General Web Browsing (`RealisticGWB`): The always-scroll-down browsing pattern in `GWB` does not reflect a realistic browsing pattern. In order to model a more realistic user web browsing behavior, the home page for each web page is instrumented to include additional movement patterns. Specifically, the `RealisticGWB` benchmark introduces vertical up-and-down, horizontal right-and-left movements, page zoom with random delays between actions. This models the browsing pattern where users spend more time reading web contents located on specific parts of a web page and skim through the rest of the page.

Education Web Browsing (`EWB-Blackboard`): As technology advances, students today are able to use their smartphones to read course announcements and get started with assignments by accessing course websites on smartphone devices. These educational websites, however, exhibit different types of contents than those included in popular websites. Unlike general-purpose websites where web contents are more sophisticated, e.g., with images, audio/video streams, or advertisement clips, web contents on these educational websites are mostly in text or document formats, where documents are often embedded in download links. This benchmark that focuses on browsing educational websites captures this set of browsing behavior in addition to `RealisticGWB`.

BlackBoard, a popular education-oriented web platform commonly used in universities to host course materials and tools, is used to model `EWB-Blackboard`. Students access Blackboard web pages for course information, announcements, assignments, discussions, etc. In addition to the Blackboard web page browsing, `EWB-Blackboard` often involves viewing assignment documents that are not directly displayed in a web browser. For example, course assignments from Blackboard web pages are often made accessible in Portable Document File (PDF) format. This means that in-between several Blackboard web page browsing sessions, students often need to switch from web browsing to document viewing.

To understand the interaction between Blackboard web page browsing and document viewing, the behavior of first browsing through Blackboard web pages and opening an assignment file embedded as a link on the Blackboard web page is modeled.

Photo Viewing (`PhotoView`): With the increasing number of pixel counts for the camera on modern smartphones (as high as 13 mega pixels), high resolution photos are prevalent on these mobile platforms. As a result, to view high resolution photos smoothly, modern smartphones must be capable of displaying high resolution photos on the screen timely for a satisfactory user experience. To represent this class of applications, high resolution photo rendering for the Android platform is modeled using a picture viewing application: *QuickPic*. The PhotoView benchmark includes consecutive photo rendering of high resolution images, each with a resolution of 4912x3264 and is of size between 4 to 6 MB.

Video Playback (`VideoPlayback`): In addition to `PhotoView`, an important class of applications for modern smartphones is high definition video playback. With popular video sharing, users today frequently view video/movie clips on their smartphones and expect high performance delivery in this application class. VideoPlayback helps to evaluate only the rendering performance for our target mobile platform, excluding any network issues that might affect our results. The application *MX Player* is used to play a high-definition (720p) MPEG-4 video of 1 minute in length.

3.2 Platform Energy Characterization with EnergyUsageCollector

The background app, EnergyUsageCollector, was implemented by modifying the code in the file PowerUsageSummary.java of the Android Settings application. Ener-

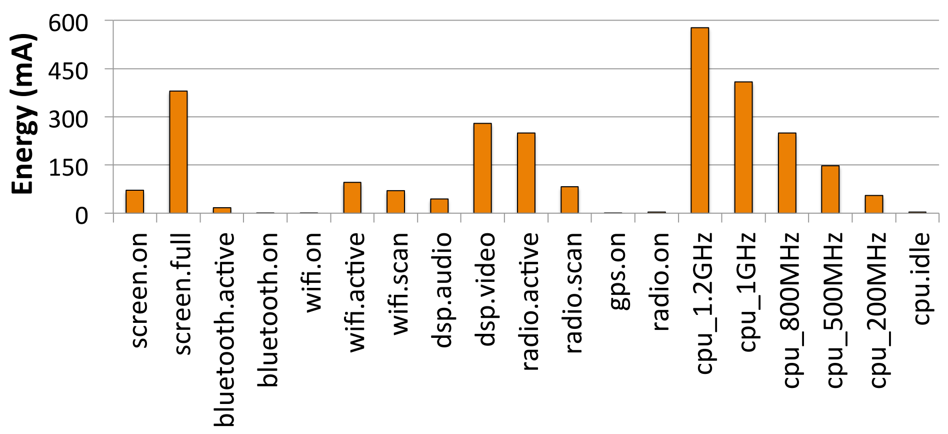


Figure 3.1: The amount of current drawn by various hardware components on Samsung Galaxy S III smartphone.

gyUsageCollector calculates the energy consumption of a running application based on two pieces of important information. First, by reading the power specification sheet (power_profile.xml in framework-res.apk) provided by smartphone vendors, the PowerUsageSummary code obtains the power consumption specific to the various hardware components. Figure 3.1 shows the amount of current drawn by various hardware components on the smartphone target, a Samsung Galaxy S III i9300. Then, EnergyUsageCollector measures the amount of time an application spends utilizing the different hardware components, e.g., application CPU cores, Wifi, the screen. The CPU power consumption takes into account the change in current draw at different frequencies, and similarly the display power consumption considers change in the brightness levels. To calculate the total energy consumed for each hardware component, EnergyUsageCollector simply multiplies the amount of time spent at each component with the power constant from power_profile.xml. For MobileBench, each application is run for a duration of 30 minutes and the application energy profile is generated. Because power_profile.xml is available in most of the modern smart-

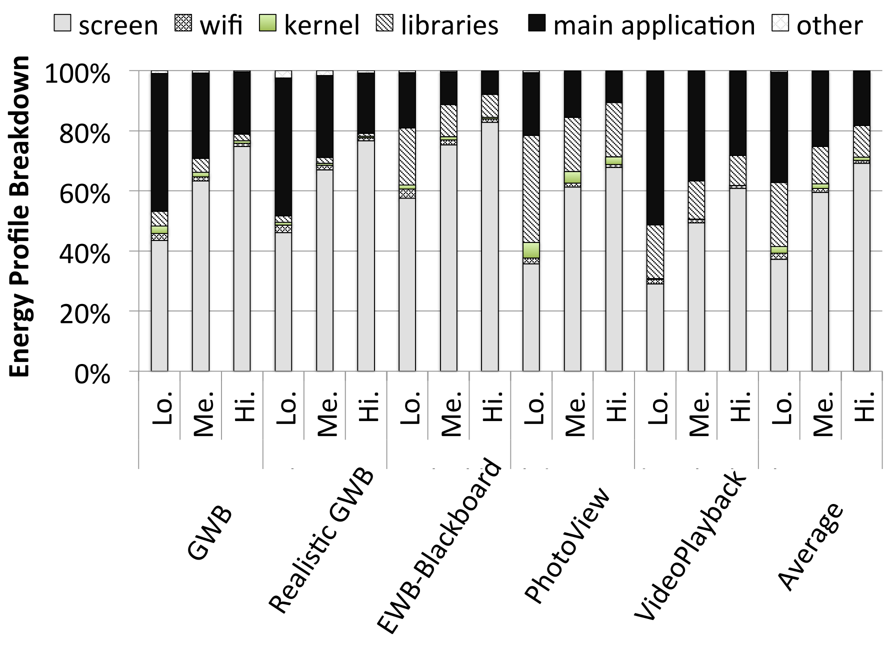


Figure 3.2: Energy profile breakdown for various smartphone components running MobileBench applications.

phones and Android uses the same code to provide battery usage information, energy profiling needs only minor code modification.

Figure 3.2 shows the energy profile for MobileBench. At the brightest level (100%), the LCD screen is undoubtedly the energy hog among all platform components. However, when the brightness of the LCD screen is lowered to 25%, the energy consumption of the application cores starts dominating.

The second important observation is that, except for general web browsing (`GWB` and `RealisticGWB`), commonly-used smartphone applications spend a significant amount of energy at executing library function calls (by as much as 36% for `PhotoView` at the screen brightness of 25% and by an average of 21% for all MobileBench applications). This is because MobileBench applications make extensive use of Android system libraries.

For media-content based applications, e.g., `Video Playback` and `PhotoView` in MobileBench, the application cores consume 30% (at 100% LCD brightness) to 70% (at 25% LCD brightness) of the total device energy. Given that users spend a significant amount of time executing media-content based applications and the auto-brightness setting is enabled on most of today's smartphone platforms (with advanced power management), the application core energy consumption becomes increasingly dominant.

Validation. The total energy consumption obtained from EnergyUsageCollector is validated with device power meter measurements. The methodology is described in detail in Chapter 6. EnergyUsageCollector estimates the energy consumption of the device with a minimum error of 3.6% and an average error of 14.5% for `VideoPlayback` and the web browsing applications. However, this error increases sharply by 3.6X for `PhotoView`; the estimated energy consumption is much higher than the meter-based measurement. One of the factors for the discrepancy is that EnergyUsageCollector does not consider the RGB components of the pixels which constitute the images. Previous study [28] has shown that the power consumption of a white pixel in comparison to that of a black pixel can be as high as 5 times. This change in energy consumption based on the color composition becomes particularly significant for `PhotoView`, which spends the majority of the time displaying the images on the screen. To improve the accuracy of EnergyUsageCollector, both EnergyUsageCollector and the default battery estimation application in the Android framework need to account for the color profile of images being displayed on the screen.

Overall, when the screen display brightness is at a reasonable, 25% brightness level, the application cores executing the main application, the library functions, and the kernel source codes, become the dominating energy-consuming component on the Samsung Galaxy S III platform. This motivates a deeper understanding of the application core's energy consumption behavior.

Chapter 4

MICROBENCHMARKS

4.1 Background

Modern smartphone processor architectures feature a hierarchical memory structure. Figure 4.1 illustrates the memory hierarchy of the ARM Cortex-A9 processor in a Samsung Galaxy S3 smartphone. Other commonly-available mobile processors, e.g., Intel Atom-based Clover Trail processors, also implement a similar memory hierarchy. When data in the memory is accessed, it will be moved across different levels of memory: from DRAM to the level-two (L2) cache, from the L2 cache to the level-one (L1) cache, and from the L1 cache to the register file. To accurately quantify the data movement energy cost, a set of microbenchmarks are designed to continuously access data in a specific level of the memory hierarchy. The data references from the microbenchmark are correlated with power readings obtained from an external power meter to compute the energy cost of each data movement operation.

4.2 Design of the Microbenchmarks

Isolating data accesses to a specific level of the memory hierarchy and quantifying the energy cost of a specific data movement are challenging in modern processors. Out-of-order execution and other important architectural optimization features, such as data prefetching and speculation, have worked well in hiding memory latencies but, at the same time, make the energy cost benchmarking for an individual instruction difficult. This necessitates the design of microbenchmarks that minimize the effect of out-of-order execution and other architectural optimizations. The design of

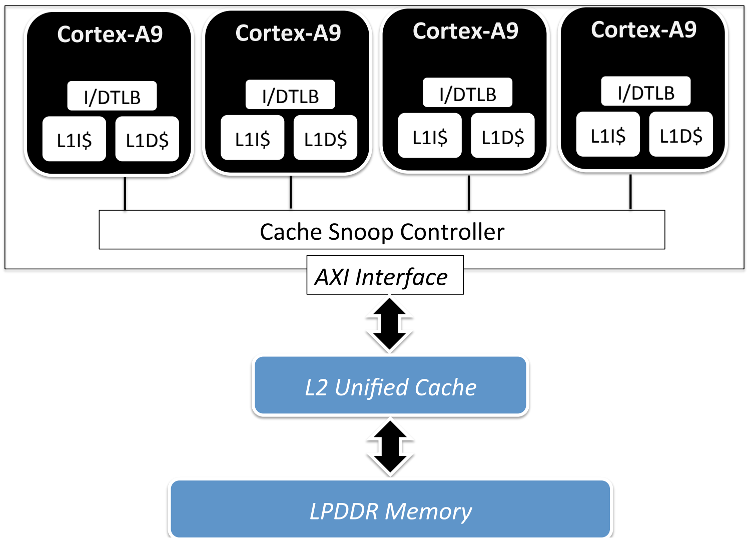


Figure 4.1: Architecture of the ARM Cortex-A9 processor in a Samsung Galaxy S3 smartphone.

the microbenchmark methodology is inspired by a recent work that quantifies the data movement energy cost for scientific applications running on desktop and server processors [24].

The goal for the benchmark design is to consistently bring data from a particular level of memory hierarchy. The program has to overcome a number of microarchitectural and compiler optimizations to accomplish that. Between the two, hardware optimizations are relatively harder to combat as they occur at runtime, are not visible to the software and only can be deduced based on performance counter values. On the other hand, compiler optimizations can be investigated by reviewing the assembly code which the compiler generates and be selectively disabled with compiler flags and appropriate programming methods. There are six different microbenchmarks that are used in this study. Four of them perform data movement operations used to compute energy costs and the remaining two are reference benchmarks. The reference benchmarks serve as a metric to understand the relative cost of data movement operations.

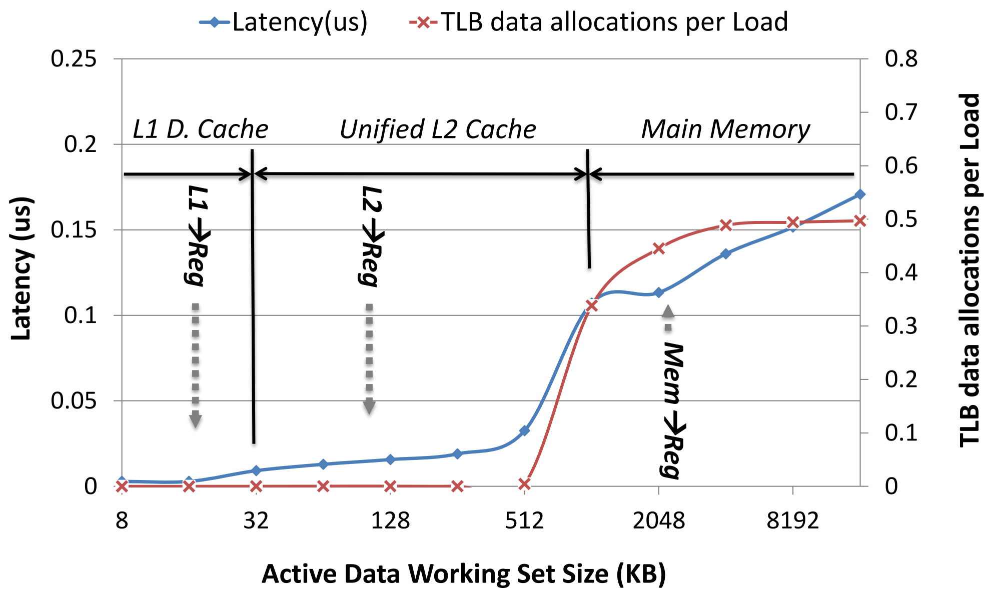


Figure 4.2: Latency measurement of the designed microbenchmarks with varying active data working set sizes.

The workings of the data movement microbenchmarks are summarized as follows:

1. Each microbenchmark first requests the operating system to allocate a memory of size that fits within the capacity of the level of memory system we are interested in.

2. The memory region is then accessed as an array of pointers. Pointer chasing is performed so that each array element access corresponds to only one architecturally executed load instruction; this avoids overhead due to array index increment. A temporary pointer variable is utilized to hold the address that refers to a word in the data set and a subsequent dereference of that pointer provides the address to another word. The sequence of addresses dereferenced can either be random or strided depending on the specific benchmark. The following snippet of C code illustrates pointer chasing.

   ```
   tmp_ptr = *(void**)tmp_ptr;
   ```

```
Initialize benchmark
Start Timer
for $i = 0$ to $i < iterations/x$ do
    .
    <operation> // repeated x times
    .
    .
end for
Stop Timer
```
Algorithm 1: Pseudo-code of the microbenchmark which performs the desired data movement in a target level of the memory hierarchy iteratively.

3. By continuously performing these dereferences in a loop, it is possible to traverse different locations within the allocated memory. The loop is timed to enable the calculation of the average latency per load instruction. The average latency is used to validate that the microbenchmark indeed performs the intended memory accesses in the targeted level of the memory hierarchy. Figure 4.2 shows the average access latency and the data TLB allocations per Load operation for different sizes of the array. There are substantial changes in the access latencies near the capacity boundaries of different levels of the memory hierarchy.

   Algorithm 1 outlines the pseudo-code of the data movement microbenchmark.

The initialization performs memory allocation for an array of pointers, calculates and writes addresses into the array for pointer chasing. If the measured average data access latency per load corresponds to the targeted level of the memory hierarchy, it can be safely assumed that the impact of operating system activities and other potentially co-executing background service processes is negligible and the designed microbenchmark performs the desired data movement patterns faithfully.

Four microbenchmarks were created to study the data movement energy across the different levels of the memory hierarchy. Each of them is listed below with description.

- **L1 cache to CPU register** ($Microbenchmark_{L1 \rightarrow Reg}$): This microbenchmark performs random accesses within the data set. Each access brings a word from the L1 data cache to the register and all data references to the L1 cache are hits. A data set size of 24KB is chosen which comfortably fits in the 32KB L1 data cache. The data access is performed using the pointer chasing logic.

- **L1 cache to CPU register w/o data dependency** ($Microbenchmark_{L1 \rightarrow Reg, no-dep.}$): The microbenchmark moves data from the L1 cache to the registers similar to $Microbenchmark_{L1 \rightarrow Reg}$, except that, for each instruction the address to be loaded from is hardcoded in the program rather than dereferenced from the previous memory access. The usefulness of this approach is explained in Chapter 5. The addresses are randomized and fit within the L1 cache.

- **L2 cache to CPU register** ($Microbenchmark_{L2 \rightarrow Reg}$): This microbenchmark performs random accesses within the data set that fits into the 1MB L2 cache of the experimental platform. The selection of the data set size for this microbenchmark requires more considerations than $Microbenchmark_{L1 \rightarrow Reg}$. This is because, additional TLB misses can be incurred by the random-walk references performed in the larger memory region. In order to ensure the majority of data movement happens between the L2 cache and the register, a data set size that fits in the 1MB L2 cache but does not fit in the 32KB L1 cache has to be chosen. This means, the L1 data cache miss rate should be as high as possible while the L2 cache miss rate is as low as possible. In addition, it is important to keep the TLB miss rate as small as possible [1] . To fulfill these constraints, the data set size for $Microbenchmark_{L2\to Reg}$ is selected to be 125KB. As shown in Figure 4.2, the selected data set size performs memory accesses that move data from the L2 cache to the register as intended since the load latency closely tracks the L2 cache access latency.

- **Main memory to CPU register ($Microbenchmark_{memory\to Reg}$):** This microbenchmark is designed to bring data from the memory to the processor register. This means that the data set size needs to be larger than the 1MB L2 cache resulting in high L1 and L2 cache miss rates. Since the data set size needs to be larger than 1MB, this microbenchmark inevitably thrashes the L2 TLB leading to page table walks. Because the additional memory references related to page table walks also access the cache hierarchy, they can lower the high L1 and L2 cache miss rates expected for the random access microbenchmark. Currently, there is not a known, effective method to differentiate cache accesses related to page table walks from those made by application load instructions. As a result, the data set size is experimentally chosen such that it has a high L2 miss rate while keeping the TLB allocations as few as possible. In this work, a data set size of 2MB is chosen. Therefore, this causes more data movement for each memory instruction on average compared to one with a smaller data set.

- **Integer and NOP ($Microbenchmark_{add}$ and $Microbenchmark_{nop}$):** In addition to the microbenchmarks which perform iterative data movement from a specific level of the memory hierarchy to the processor register, these microbenchmarks that execute integer addition and NOP instructions continuously are designed to

[1]The 128-entry TLB effectively covers the memory region of 512KB

understand their relative impact on energy consumption with respect to that of data movement.

4.3 Discussion

There are other challenges and considerations in the process of designing and running the microbenchmarks, which are discussed below.

- **The Influence of TLB:** Continuously accessing words within a data set whose working set is larger than the address space covered by the L1 TLB and L2 TLB results in TLB misses. The Cortex A9 performs hardware page table walks on a L2 TLB miss and this results in two additional memory references with two-level page table. As the page table can be cached in the Cortex A9, these references due to page table walk alter the high L1 and L2 miss rates the random access microbenchmark is designed to produce. Furthermore, without the performance counters events to precisely differentiate between L1 data cache references made by the program and references related to the TLB miss handler, we are left with a lower overall L1 and L2 miss rate. Therefore, a data set size that has a higher L2 miss rate while keeping the TLB allocations to a minimum is chosen. Therefore, the data set size is chosen such that the TLB misses are lower.

- **Compiler Optimization:** To verify that the designated data access pattern is not affected by compiler optimizations, the assembly code generated for the microbenchmarks is reviewed. For example, the variable that holds the temporary pointer could be removed by the compiler.

- **Loop Unrolling Effect:** The main loop of memory accesses in the microbenchmarks is unrolled such that a larger number of memory loads are executed for each loop iteration to reduce the frequency of branch instruction execution. The loop size/iterations is carefully selected such that the loop body is large enough for the reduced overhead of loop index increment and branch instructions but not too large to cause unintended, additional instruction cache misses.

- **Context Switch and OS Scheduling Overhead:** The loop count should be large enough for the benchmark to run for a sufficiently long period of time so that the power measurement device can collect enough samples for analysis, but not too large such that the operating system/other programs could affect the execution time of the benchmark with context switch overhead. Since, the benchmarks are run on a commercial device which is not tailored for benchmarking, the possibility of such interference is likely if not cared for.

- **Priority Setting and Task Migration:** To minimize interference with other running processes, the microbenchmarks are given the highest priority by setting the `nice` number [2] to -20. Furthermore, to prevent task migration between the different, available cores, the microbenchmarks is pinned to a specific core at the beginning of the program execution.

- **Frequency:** The microbenchmark's power consumption highly depends on the core frequency setting, which the default Android/Linux operating system varies dynamically based on runtime core utilization. To eliminate the influence of frequency variation, the *performance* CPU governor is selected to set the frequency of the cores to 1.4 GHz.

- **Spatial Locality:** Finally, spatial locality needs to be taken into account in the microbenchmark design. To ensure that data accesses always result in cache misses, only one data word within a cache line is accessed.

[2]`nice` is a Linux program to give a process more or less CPU time than other processes. A niceness of -20 is the highest priority and 19 is the lowest priority.

Chapter 5

DATA MOVEMENT ENERGY COST EVALUATION

This chapter describes the techniques that are used to measure the energy cost of individual instructions and to isolate the components that do not contribute to the data movement energy, e.g., idle energy and stalled cycles. Figure 5.1 illustrates the approach to determining the energy cost for accessing data in different levels of the memory hierarchy. The energy cost for moving data from the L1 cache to the registers ($\Delta Energy_{L1}$) is first determined. This process is repeated to determine the energy cost for moving data from the L2 cache to the L1 cache ($\Delta Energy_{L2}$) and from the memory to the L2 cache ($\Delta Energy_{mem}$). Since the number of instructions performed in the body loop of the microbenchmarks is in the order of billions, the majority of the processor energy consumption is spent on the designed data movement.

5.1 Dynamic Energy Measurement for the Microbenchmarks

The scope for external power measurements with commercial smartphones is limited significantly compared to that of product development boards, which often come with exposed test points/voltage rails. The only power measurement access points offered in our test device are the battery terminals. The entire system, consisting of the display, LEDs, speakers, DRAM, SoC, sensors, eMMC etc. is powered via this set of terminals. For the data movement power measurements at this terminal to be meaningful, all the peripheral components are turned off or kept inactive through options that the OS provides. When the display, Wi-Fi, mobile radio and other peripherals are turned off, the application processor and DRAM consume most of the power since the microbenchmarks used in the measurements do not make use any

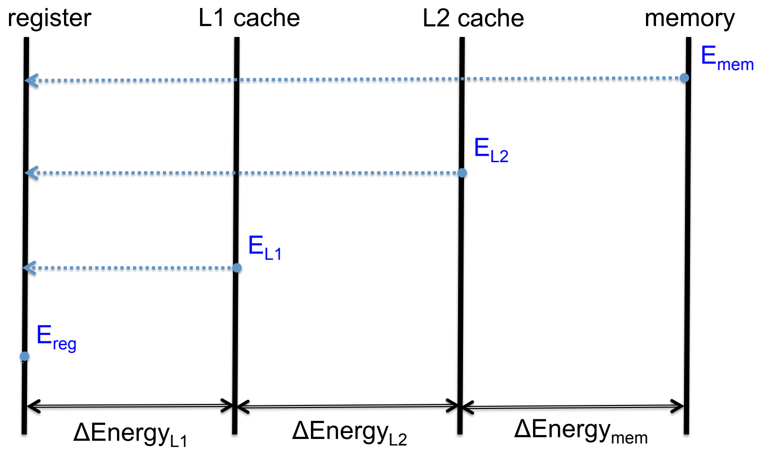


Figure 5.1: Energy measurement for data movement across the memory hierarchy.

peripheral components. To minimize potential interference, unnecessary background service processes that can potentially skew measurements of the microbenchmarks are manually terminated. The power measurements are made in stable memory access phases (regions of interest) when the program is fully loaded into memory and executes only the desired data movement instructions. The microbenchmarks are written in C, cross-compiled on the host machine with the ARM-Android NDK tool chain [3]. The binaries are then pushed to the device to be launched from the host machine via the Android Debug Bridge (adb) terminal. The L2 prefetcher is turned on/off while evaluating the energy costs for microbenchmarks by modifying a part of architecture-specific kernel start-up code. The Linux kernel modules in [1] are extended to read the L2 cache controller registers and to validate the configuration that has been set by the kernel.

In order to calculate the power consumption of the microbenchmarks, the baseline power or the **idle power** consumption of the device before the microbenchmarks begin to execute is recorded and subtracted from the measured value. Thus, the power consumption attributed to the microbenchmarks is

$$P_{microbenchmark} = P_{device} - P_{idle}$$

Furthermore, the total energy consumption of the microbenchmarks is

$$E_{microbenchmark} = \int_{StartTime}^{EndTime} P_{microbenchmark} \mathrm{d}t$$

5.2 Stall Cycles

Depending on where data resides, it takes from 4 to 200 cycles to bring the requested data to processor registers. This means that the processor could spend a majority of time waiting for data, resulting in significant stall cycles. The stall cycles increase when the memory instruction in an application depends on the data requested by the previous instruction, e.g., in the pointer-chasing microbenchmarks.

In order to separate the energy cost of stall cycles from the energy cost of moving data from one level to another level of the memory hierarchy, a matching microbenchmark, $Microbenchmark_{L1\rightarrow Reg,no-dep.}$ is utilized. This benchmark performs exactly the same data movement as in $Microbenchmark_{L1\rightarrow Reg}$, except that all data dependencies in the original microbenchmark have been removed. Since all memory addresses in $Microbenchmark_{L1\rightarrow Reg,no-dep.}$ are known, the memory accesses are independent of each other and, thus, the stall cycles caused by data dependency is removed. The number of stall cycles in $Microbenchmark_{L1\rightarrow Reg}$ is obtained from the performance counters.

By comparing the energy consumption of $Microbenchmark_{L1\rightarrow Reg,no-dep.}$ with the pointer chasing $Microbenchmark_{L1\rightarrow Reg}$, we can compute the stall cycle energy consumption. Using values measured from the hardware performance counters, it can be deduced that $Microbenchmark_{L1\rightarrow Reg}$ creates three pipeline stalls for each load that is issued. Therefore, Stall Cycle energy can be computed as:

$$E_{Stall} = (E_{L1toReg\rightarrow Reg} - E_{L1toReg\rightarrow Reg,no-dep})/N_{Stalls}$$

5.3 Long Latency Memory Operations

The $Microbenchmark_{L2\rightarrow Reg}$ and $Microbenchmark_{RAM\rightarrow Reg}$ benchmarks with the pointer chasing logic like $Microbenchmark_{L1\rightarrow Reg}$ cause several stall cycles due to the data dependency between loads. The stall cycle energy has to be subtracted from the energy measurement values for both benchmarks to isolate the data movement energy. The same formula is used for these long latency operations.

$$E_{L2\rightarrow Reg} = (E_{L2\rightarrow Reg} - E_{Stall} * N_{Stalls})/N_{MemoryAccesses}$$

5.4 Cache Prefetcher

Hardware prefetching is a commonly-used latency mitigation technique in modern processors. Cache prefetchers bring data into the cache hierarchy before the actual reference, thereby shortening the memory latency of application demand requests. While often helpful, the benefits of aggressive prefetching hinge on its accuracy. When effective, memory performance can be significantly improved. However, inaccurate or untimely prefetched cache lines can result in additional data to be brought into the cache, which amounts to wasted energy. As energy is a key limited resource in mobile platforms, it is of critical importance to evaluate the energy cost of prefetching. We approximate this energy cost of prefetching as $E_{RAM\rightarrow L2}$, i.e., same as moving data from the memory to L2 cache. The rationale is that the energy cost of prefetching a line from memory is mostly expended on the actual data movement [24]. Isolating the energy consumption of prefetch engine's overhead from this is not apparent, due to a combination of the energy measurement methodology that is adopted and the limited access to component level power measurement on a production smartphone.

There are three distinct prefetchers on the Samsung Exynos-based SoC which houses the ARM Cortex A9 processor: per-core L1 cache stride prefetchers, L2 double line-fill cache prefetcher, and the L2 cache stride prefetcher [5].

The per-core L1 cache prefetchers monitor cache references to the L1 cache based on the program counter (PC) value and address and are capable of tracking multiple prefetch streams. The L1 cache prefetchers bring data from the lower levels of the cache hierarchy in advance by placing the prefetched cache lines into a dedicated prefetch buffer. Upon hits, prefetched data are brought from the prefetch buffers to the L1 caches. In the case of inaccurate prefetch requests, the prefetcher throttles down its aggressiveness to reduce the degree of potential interference in the prefetch buffer. Apart from this, the L1 prefetcher also sends prefetch hints to the L2 cache controller for prefetching lines into the L2 cache. These lines that are allocated in the L2 cache are not sent to the L1 cache. The L2 double line-fill cache prefetcher observes the L2 cache misses and fetches two cache lines – the one that caused a miss and the next line from the memory. The L2 controller implements stride prefetching mechanism that fetches a pre-configured number of cache lines based on the references it receives.

Chapter 6

EXPERIMENTAL SETUP

This chapter introduces the experimental methodology for real-system energy measurements and provides the background to the measurement infrastructure this work makes use of.

6.1 Experimental Platform

All experiments presented are performed on a Samsung Galaxy S3 I9300 smartphone which houses the Samsung-made Exynos4 Quad 4412 SoC. The SoC has four Cortex-A9 application cores and an integrated L2 cache. The device runs a rooted Cyanogenmod that is based on Android 4.3. This specific model ships with a 1GB Low Power DDR (LPDDR) memory. The relevant technical specifications are provided in below table 6.1.

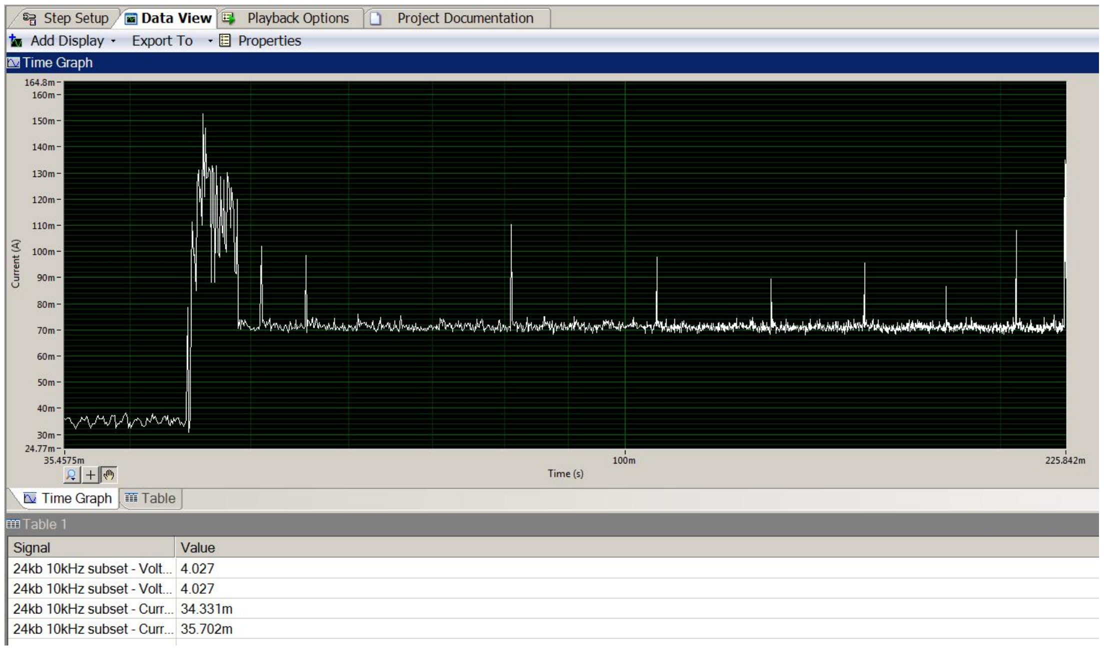
6.2 Energy Measurement for the Experimental Platform

The experiments presented in this thesis rely on the described power consumption measurements of the smartphone device. To perform power measurements, the Li-Ion battery is removed and the device is powered with a DC power supply set to 4.0V. The first application of energy measurements is the validation of energy consumption estimate generated by EnergyUsageCollector 3.2. The device energy consumption is measured with a Watt's Up RC Watt meter [15] that has a current measurement resolution of 0.01A. The source side of the meter is connected to Power Supply with the load side connected to the smartphone. The measured values are manually recorded for comparison. In contrast, the data movement energy measurements described in

Table 6.1: Technical Specifications of the System.

Operating System	Android Jelly Bean 4.3
Display	Super AMOLED with capacitive touchscreen
Display Size	720 x 1280 pixels, 4.8 inches
ISA	ARMv7 processor architecture
CPU	4 ARM Cortex A9 cores
TLB	L1 I/DTLB 32-entries, full-assoc.; L2 TLB 2-way set-assoc. 128 entries
L1 Inst. Cache	32KB, 4-way, Private
L1 Data Cache	32KB, 8-way, Private
L2 Unified Cache	1MB, 16-way, Shared, Inclusive
Main Memory	1GB
Page size	4KB

5 are performed with the National Instruments DAQ 6251 [20]. The DAQ offers a higher resolution and sampling rate for measurements compared to the Watt's Up power meter . In this case, the power supply is connected to the battery terminals by a pair of test clips and a small shunt resistor in between to form the measurement circuit. The DAQ periodically samples voltages across the resistor that is then used to calculate the current flow through the circuit. The readings are displayed in the NI SignalExpress [21] tool installed on a host NI PXI Controller. Figure 6.1 is a captured snapshot of the time graph that shows the current and voltage. SignalExpress has several features that aid in data recording, run-time calculation of power and post-measurement analysis. Further, the data logged by the DAQ is minimally processed to eliminate noise by time averaging, histogram analysis and DC component extraction. Both current and voltage are sampled at 100KHz with a resolution of 10^{-6}. All data movement energy measurement results presented in this thesis are obtained with the lowest brightness setting for the display.



Signal	Value
24kb 10kHz subset - Volt...	4.027
24kb 10kHz subset - Volt...	4.027
24kb 10kHz subset - Curr...	34.331m
24kb 10kHz subset - Curr...	35.702m

Figure 6.1: Current and Voltage measurement with NI SignalExpress

6.3 Performance Counters

The Linux kernel that runs on the device is configured at build time to enable profiling, tracing, high resolution timer support and access to performance counters. This configuration is necessary to enable performance counter reading in the CPU and the L2 cache controller with ARM Streamline [6]. Streamline is a CPU profiling and performance analysis tool that runs on a host machine and connects to a target device through adb. This setup establishes adb connection between the host and the target via the micro USB connection on the smartphone. Streamline provides a graphical timeline view of a selection of statistics from the Linux kernel's sysfs virtual file system and the processor's Performance Monitoring Unit(PMU). Figure 6.2 is a snapshot of Streamline's timeline with PMU statistics captured. The PMU counters are sampled and the readings are streamed to Streamline by the driver *gator* and user space daemon *gatord*, both running on the target. Gator is built as a Linux

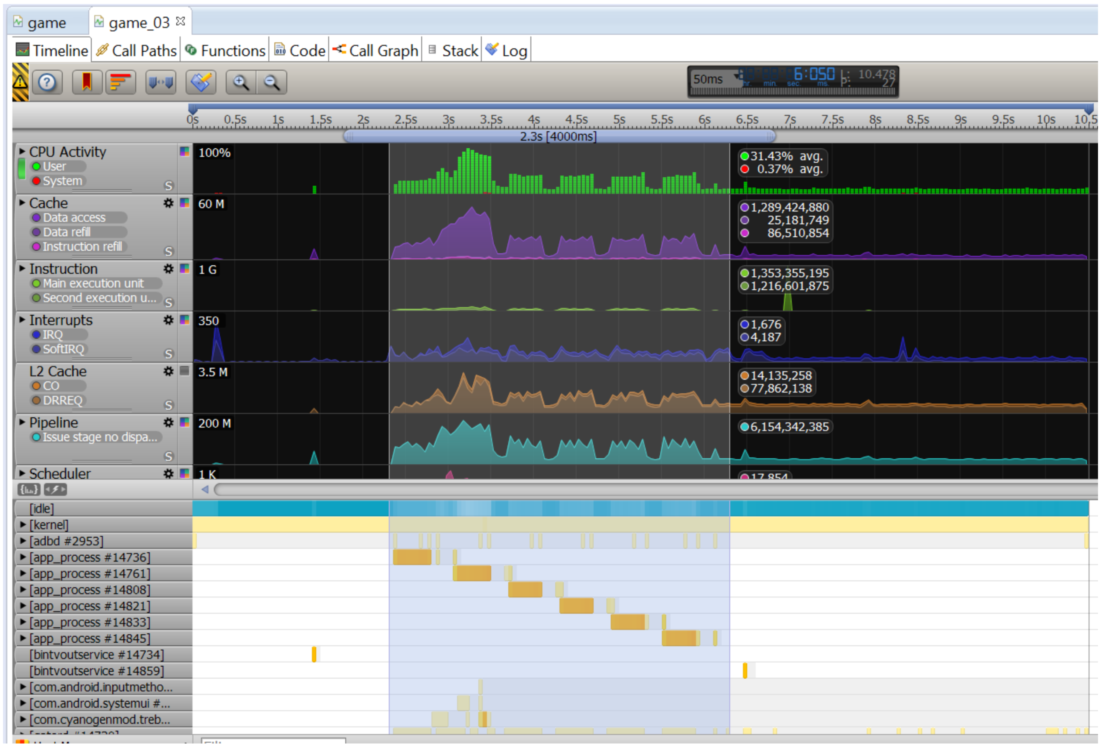


Figure 6.2: Timeline view of performance couner statistics from ARM Streamline

kernel module using the Android NDK toolchain. The driver configures the PMU registers to measure the performance events specific to the architecture and samples the registers that hold the event counts. Similarly, the L2 cache controller that is integrated with the application cores provides two counters. The events that Cortex A9 architecture supports and are of interest to this work are L1 data cache misses, L1 data cache accesses, L1 instruction cache misses, total instructions executed and the number of Load, Store and Integer instructions. Similarly, the L2 cache controller can record the number of L2 reads, L2 cast outs, L2 writes etc.

Chapter 7

DATA MOVEMENT CHARACTERIZATION FOR MOBILE WORKLOADS

Table 7.1 summarizes the energy costs for each of the operations described in Chapter 5. The methodology that has been developed so far is now leveraged to analyze the impact of data movement for real world, smartphone applications. The total energy consumption measurement for a diverse set of mobile workloads, e.g., the MobileBench suite and a game workload, FrozenBubbles, from the Moby suite [19], is obtained by sampling the dynamic power consumption, P_i. The energy consumption is calculated by integrating the power samples over time with the trapezoidal rule.

$$Energy = \int_{StartTime}^{EndTime} P_t \mathrm{d}t = \sum_{i=0}^{k} P_t t_i$$

The energy consumption of the application is the difference between the total device energy consumption and the idle energy consumption. The data movement energy is estimated by multiplying the number of accesses to the L1, L2 caches, and the main memory with the respective unit energy costs in table 7.1 for moving data between the different levels. Similarly, the energy spent on processor stall cycles is also evaluated. By separating the data movement and stall cycle energy consumption from the total energy consumption of the experimental device, we can attribute the rest of the energy consumption to the application processor, other SoC accelerators which may be active and performing computations concurrently with the application processor, other system peripherals (e.g., SD card access), as well as the display.

Figure 7.1 shows the energy breakdown for the mobile workloads. The *Data Movement Energy* bars represent the portion of the total device energy consumption due to the data movement in the application processor's memory hierarchy and the *Stall*

Table 7.1: Energy Cost of Data Movement.

Operation	Energy Cost (nJ)	Δ Energy (nJ)	Equivalent ADD Ops.
NOP	0.105	-	-
ADD	0.105	-	1
LOAD L1→ Reg	0.192	0.192	1.83
LOAD L2→ Reg	0.803	0.611	7.65
LOAD DRAM→ Reg	12.032	11.228	114.6
Stall cycle	0.068	-	-

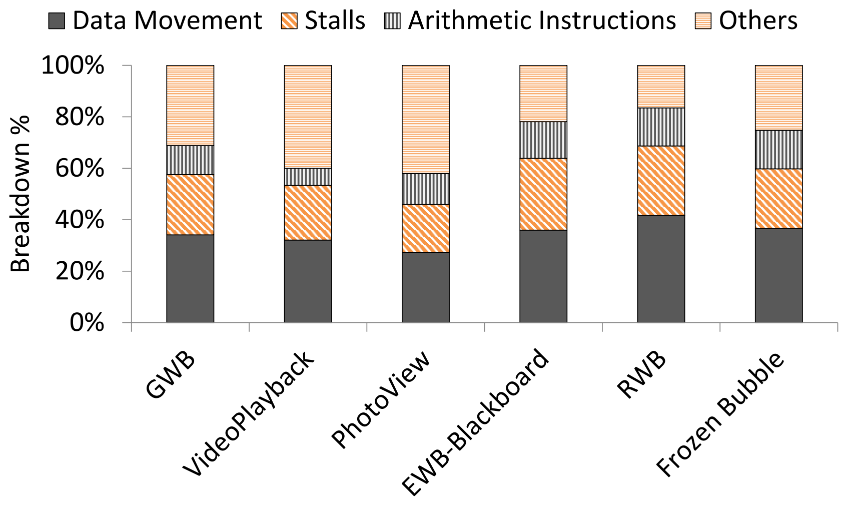


Figure 7.1: Energy breakdown for the experimental device.

Cycle Energy bars represent the portion of the total device energy consumption from processor stall cycles. The energy consumption due to the execution of arithmetic instructions is represented by the portion labeled *Arithmetic Instructions*. The *Others* bars represent the rest of the energy consumption of other active components in the device. On average, a significant portion (34.6%) of the total device energy consumption is spent on moving data from one level of the memory hierarchy to the next level for mobile workloads. The data movement energy is particularly high (41%) for realistic web browsing (`RWB`). Relatively `PhotoView` spends less amount of energy in data movement. This is likely due to the application using hardware acceleration for *jpeg* decoding. As a result, more energy is spent on the *Others* category for `PhotoView`.

Another interesting observation is that there is a considerable amount of energy spent on stall cycles in the application processor. On average, 23.5% of the total device energy is spent on stalled cycles, e.g., resolving data dependencies, waiting for long latency memory operations, etc. This stall cycle energy is expected to increase considering realistic user behavior for mobile devices. Users typically do not user their smartphones for continuous computations. Typical smartphone usage reveals a pattern of a short-term use, e.g., texting, viewing pictures, searching for restaurants, followed by a long period of idle time. While today's Android OS already adopts smart energy management policies that aggressively modulate down the operating frequency of the application processor or even puts the application processor into the sleep mode, the stalled cycle energy in the processor cannot be completely eliminated by such coarse-grained energy management. This urges architects for mobile processors to integrate more, but simplified, cores into the application processor to reduce the energy cost of stall cycles which can translate to improved energy efficiency. Finally, the energy cost of computations, hardware accelerators, etc., in the *Other* category varies from 31.3% to 54.1%. A significant portion of the *Other* energy consumption comes from the smartphone display, which has been shown as one of the most power-hungry components in modern smartphones [13, 33].

In addition to the energy breakdown, the energy analysis for data moving from one level of the memory hierarchy to another level is also performed. Figure 7.2 shows the relative energy cost for moving data from the L1 cache to processor register ($L1 \rightarrow Reg$), from the L2 cache to the L1 instruction cache ($L2 \rightarrow L1Instruction$), from the L2 cache to the L1 data cache ($L2 \rightarrow L1Data$), from the memory to the L2 cache by the processor ($Mem \rightarrow L2$) and by the cache prefetchers ($Prefetches$). Depending

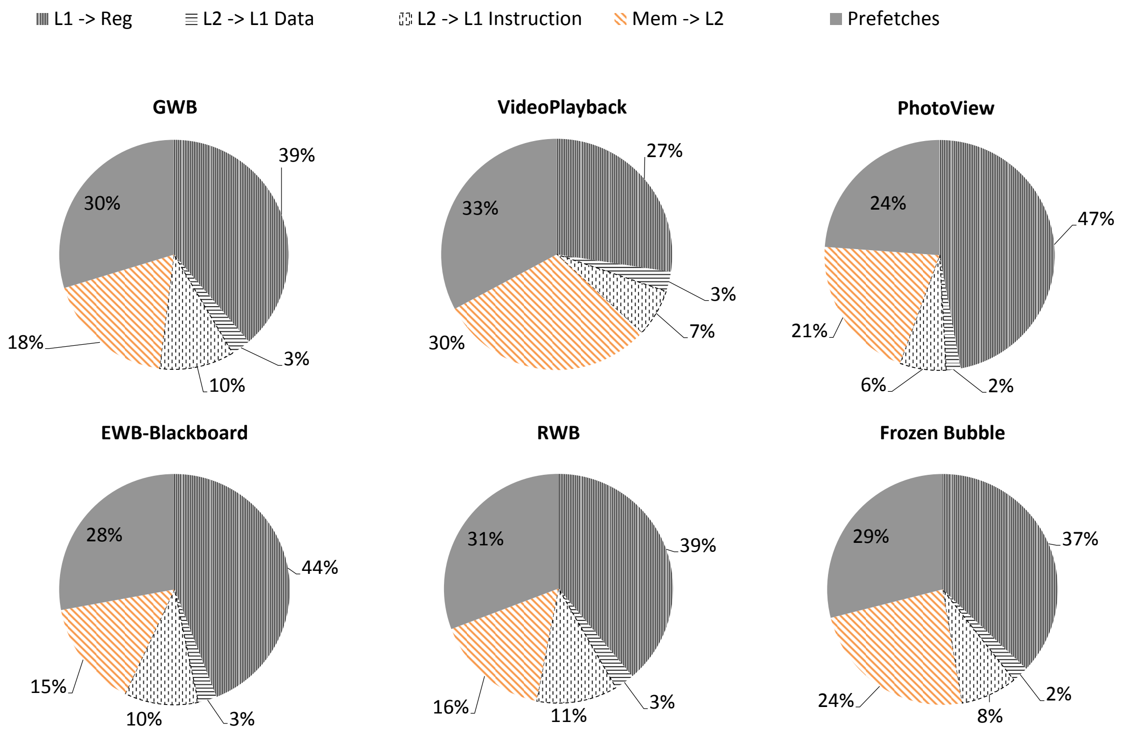


Figure 7.2: Relative energy cost for moving data from one level to another level of the entire memory hierarchy in the ARM Cortex-A9 processors for mobile workloads.

on the memory access patterns of the mobile workloads, the energy consumption dedicated to each level of the memory hierarchy varies. For all studied workloads except for `VideoPlayback`, $L1 \rightarrow Reg$ is the most significant. The reason for the relatively lower $L1 \rightarrow Reg$ for `VideoPlayback` is because the active working set of this benchmark does not completely fit in the cache hierarchy, having a higher L2 cache miss rate of 24.86%. Thus, a considerable amount of energy is spent moving data from the memory to the L2 cache.

Another interesting observation for the studied mobile workloads is the relatively higher data movement energy to bring instructions to the L1 instruction cache than to bring data to the L1 data cache. This is because mobile workloads often heavily rely on built-in libraries and system calls in the OS (Android Jelly Bean 4.3 in this case) and thus exhibit larger instruction working sets that can exceed the size of the L1 instruction cache. This is similarly shown in an older study by Guitierrez et al. [16]. Overall, the data movement energy of $Mem \rightarrow L2$ is dominating. This motivates mobile processor and SoC architects to optimize the data path between the memory and the L2 cache, which will translate to significant energy consumption reduction and improved energy efficiency gains.

Chapter 8

CONCLUSION

8.1 Summary of Results

This thesis describes a detailed methodology for quantifying the data movement energy cost on a commercial smartphone and summarizes the energy cost for moving data from one to another level of the memory hierarchy in a mobile processor, ARM Cortex-A9. With the instruction-level energy cost quantification, detailed characterization is presented for the portion of the energy that is spent on data movement for a diverse set of smartphone applications. Overall, the energy spent on data movement in mobile processors is significant. 34.6% of the total device energy consumption is spent on data movement. The data movement energy is particularly high (41%) for realistic web browsing that is commonly performed on smartphones. The results also indicate a relatively high stalled cycle energy consumption (an average of 23.5%) for current smartphones. With the experimental methodology, detailed energy characterization and insights provided, this thesis lays a foundation to further explore data movement energy cost constraints in smartphone SoCs.

8.2 Future Research Directions

8.2.1 Smartphone processors

The substantial contribution by the stall cycle component to total power consumption motivates the design for future mobile processors with more but simpler cores. Simpler cores can reduce the energy cost per stall cycle. Furthermore, the considerable amount of energy spent on moving data from the memory to the L2 cache encourages more research into low-power emerging memory technologies for embedded devices.

8.2.2 Data movement in heterogeneous mobile systems

Increasingly, GPUs with their massively parallel architecture are being exercised to accelerate highly parallel kernels within applications. The introduction of the Nvidia CUDA [29] programming model has simplified the programming interfaces, letting programmers leverage the potential of GPUs to perform non-graphics computations. Another distinct approach to program GPUs is the OpenCL [31] programming language, which has enabled developers to write cross-platform kernels that are portable to different architectures. These kernels can either be compiled offline to a device or JIT compiled at run-time to run on any supporting device. Although, both approaches are fairly well-established in the context of desktop and server computing environments, the same cannot be said with reference to mobile computing. But, with emerging trends in mobile architectures leaning towards heterogeneous designs, suitable programming models are also evolving. The recent Samsung SoCs for mobile devices like the Exynos 5420 [36] and 5422 [37] have programmable GPUs that allow developers to write OpenCL or Renderscript [4] programs for general purpose computations and offload the computations to the GPU. Likewise, the latest Snapdragon SoCs [35] from Qualcomm have Adreno GPUs that are capable of supporting general computations aside from enabling graphics. The changes in hardware along with the availability of corresponding run-time libraries will allow developers write device agnostic code where the target can be CPU, GPU or a DSP. Also, in mobile SoC's where typically the GPU and CPU share system memory, the scope for data movement characterization is even wider. To illustrate the idea, a preliminary investigation was carried out to quantify the data movement energy cost between the GPU and memory.

Different from the characterization presented in the rest of this thesis, the target platform for this study was the Arndale Octa development board [10] that has Exynos 5420 SoC. The primary reason being, the Mali-400 MP [7] GPU in the Exynos 4412 does not support general purpose computing, whereas, the Mali-T628 [8] GPU cores in the Exynos 5420 explicitly support the OpenCL 1.1 framework. An OpenCL microbenchmark was developed using the Mali SDK [9] to generate strided memory accesses within a large data set that does not fit within the GPU cache. The idea is to repeatedly move data from the shared system memory to the GPU core. The power consumption of the board is recorded during the idle phase as well as the memory access phase to compute the benchmark's power consumption. Each memory access from the memory to the GPU core is estimated to cost 370.1 nJ, which is several times larger than the energy cost reported for the CPU counterpart in Chapter 7. This huge difference is likely due to the stall cycle energy cost that is included in the estimated 370.1 nJ energy consumption. Another benchmark that performs continuous ADD operations is designed and executed on the GPU. The corresponding energy cost is evaluated to be 0.3179 nJ. This energy consumption for performing an ADD operation on the GPU is also much higher than that of the CPU. We speculate this higher energy cost to come from the GPU arithmetic logic that is possibly less-optimized for general-purpose compute. Notwithstanding the absence of an exact analysis, this experiment stimulates the need to explore the role of data movement in heterogeneous architectures like the mobile SoC. A thorough analysis will be immensely useful to make run-time decisions with regards to choosing the target platform for execution.

8.2.3 Data movement in client-server environments

A growing number of mobile applications, including email, word processing, media players, etc., offload parts of their computation, in varying degrees, to the cloud. In most cases, the smartphone app itself is a light-weight client that relies on the remote servers in cloud to perform heavy-duty computations. Kestor et al. through their detailed analysis in [24] shed light on the relative costs of data movement in processors targeted at servers. By taking into account the energy consumption incurred in transmitting data through the network along with the energy spent in computations at the server-end, it is possible to deduce the total energy consumption involved. This figure can then be compared against performing computations in the smartphone itself. Characterizing the energy consumption involved and studying such trade-offs from both energy and performance points of view will offer new metrics that can be used by developers to distribute application execution.

REFERENCES

[1] Cortex a9 prefetch disable. URL https://github.com/deater/uarch-configure/tree/master/cortex-a9-prefetch.

[2] S Amarasinghe, M Hall, R Lethin, K Pingali, D Quinlan, V Sarkar, J Shlf, R Lucas, K Yelick, P Balaji, P. C. Diniz, A Koniges, M Snir, and S. R. Sachs. Report of the workshop on exascale programming challenges. In *Technical report, US Department of Energy*, 2011.

[3] Android. Ndk, . URL https://developer.android.com/tools/sdk/ndk/index.html.

[4] Android. Renderscript, . URL http://developer.android.com/guide/topics/renderscript/compute.html.

[5] ARM. Pl310 cache controller technical reference manual, . URL http://goo.gl/GUP7xf.

[6] ARM. Streamline analyzer, . URL http://ds.arm.com/ds-5/optimize/.

[7] ARM. Mali-400 mp, . URL http://www.arm.com/products/multimedia/mali-cost-efficient-graphics/mali-400-mp.php.

[8] ARM. Mali-t628, . URL http://www.arm.com/products/multimedia/mali-performance-efficient-graphics/mali-t628.php.

[9] ARM. Mali opencl sdk, . URL http://malideveloper.arm.com/develop-for-mali/sdks/mali-opencl-sdk/.

[10] Arndale. Octa. URL http://www.arndaleboard.org/wiki/index.php/Introduction.

[11] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[12] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 2000.

[13] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of USENIX annual technical conference (USENIX-ATC)*, 2010.

[14] Bill Dally. Power, programmability, and granularity: The challenges of ExaScale computing. In *Proceedings of International Parallel and Distributed Processing Symposium*, 2011.

[15] RC electronics. Watt's up. URL `http://www.rc-electronics-usa.com/ammeters/dc-amp-meter.html`.

[16] A Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver. Full-System Analysis and Characterization of Interactive Smartphone Applications. In *Proc. of the International Symposium on Workload Characterization*, 2011.

[17] Shuai Hao, Ding Li, William G. J. Halfond, and Ramesh Govindan. Estimating mobile application energy consumption using program analysis. In *Proceedings of the 2013 International Conference on Software Engineering*, 2013.

[18] John L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 2006.

[19] Yongbing Huang, Zhongbin Zha, Mingyu Chen, and Lixin Zhang. Moby: A mobile benchmark suite for architectural simulators. In *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, 2014.

[20] National Instruments. Pxi-6251 daq, . URL `http://sine.ni.com/nips/cds/view/p/lang/en/nid/14125`.

[21] National Instruments. Signalexpress, . URL `http://www.ni.com/labview/signalexpress`.

[22] Nvidia Tegra K1. URL `http://www.nvidia.com/object/tegra-k1-processor.html`.

[23] Steve Keckler. Life after dennard and how i learned to love the picojoule (keynote speech). In *Proceedings of International Symposium on Microarchitecture*, 2011.

[24] G. Kestor, R. Gioiosa, D.J. Kerbyson, and A. Hoisie. Quantifying the energy cost of data movement in scientific applications. In *Proceedings of International Symposium on Workload Characterization*, 2013.

[25] Sheayun Lee, Andreas Ermedahl, Sang Lyul Min, and Naehyuck Chang. An accurate instruction-level energy consumption model for embedded risc processors. In *Proceedings of the ACM SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems*, 2001.

[26] Sheng Li, Jung-Ho Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, 2009.

[27] Microsoft Office Mobile. URL `https://play.google.com/store/apps/details?id=com.microsoft.office.officehub&hl=en`.

[28] R. Murmuria, Jeffrey Medsger, A. Stavrou, and J.M. Voas. Mobile application and device power usage measurements. In *Proceedings of International Conference on Software Security and Reliability*, 2012.

[29] Nvidia. Cuda. URL http://www.nvidia.com/object/cuda_home_new.html.

[30] Samsung Exynos Octa. URL http://www.samsung.com/global/business/semiconductor/product/application/detail?productId=7977&iaId=2341.

[31] OpenCL. URL https://www.khronos.org/opencl/.

[32] Dhinakaran Pandiyan and Carole-Jean Wu. Quantifying the energy cost of data movement for emerging smartphone workloads on mobile platforms. In *Proceedings of International Symposium on Workload Characterization*, 2014.

[33] Dhinakaran Pandiyan, Shin-Ying Lee, and Carole-Jean Wu. Performance, energy characterization and architectural implications of an emerging mobile platform benchmark suite – MobileBench. In *Proceedings of International Symposium on Workload Characterization*, 2013.

[34] Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of European Conference on Computer Systems*, 2011.

[35] Qualcomm. Snapdragon 810. URL https://www.qualcomm.com/products/snapdragon/processors/810.

[36] Samsung. Exynos 5420, . URL http://www.samsung.com/global/business/semiconductor/product/application/detail?productId=7977&iaId=2341.

[37] Samsung. Exynos 5422, . URL http://www.samsung.com/global/business/semiconductor/product/application/detail?productId=7978&iaId=2341.

[38] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the International Symposium on Microarchitecture*, 2009.

[39] Amit Sinha, Nathan Ickes, and Anantha P. Chandrakasan. Instruction level and operating system profiling for energy exposed software. *IEEE Trans. Very Large Scale Integr. Syst.*, 2003.

[40] International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker, . URL http://www.idc.com/getdoc.jsp?containerId=prUS24645514.

[41] International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker, . URL http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22689111.

[42] International Data Corporation (IDC) Worldwide Quarterly PC Tracker, . URL http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22653511.

[43] The Verge. Apple announces 1 million apps in the app store, more than 1 billion songs played on itunes radio. URL http://www.theverge.com/2013/10/22/4866302/apple-announces-1-million-apps-in-the-app-store.

[44] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of CODES+ISSS*, 2010.