Quantifying the Energy Cost of Data Movement for Emerging Smart Phone Workloads on Mobile Platforms

Dhinakaran Pandiyan and Carole-Jean Wu
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, Arizona 85281
Email: {dpandiya,carole-jean.wu}@asu.edu

***Abstract*—In portable computing systems like smartphones, energy is generally a key but limited resource where application cores have been proven to consume a significant part of it. To understand the characteristics of the energy consumption, in this paper, we focus our attention on the portion of energy that is spent to move data to the application core's internal registers from the memory system. The primary motivation for this focus comes from the relatively higher energy cost associated with a data movement instruction compared to that of an arithmetic instruction. Another important factor is the distributive computing nature among different units in a SoC which leads to a higher data movement to/from the application cores.**

We perform a detailed investigation to quantify the impact of data movement on overall energy consumption of a popular, commercially-available smart phone device. To aid this study, we design micro-benchmarks that generate desired data movement patterns between different levels of the memory hierarchy and measure the instantaneous power consumed by the device when running these micro-benchmarks. We extensively make use of hardware performance counters to validate the micro-benchmarks and to characterize the energy consumed in moving data. We take a step further to utilize this calculated energy cost of data movement to characterize the portion of energy that an application spends in moving data for a wide range of popular smart phone workloads. We find that a considerable amount of total device energy is spent in data movement (an average of 35% of the total device energy). Our results also indicate a relatively high stalled cycle energy consumption (an average of 23.5%) for current smart phones. To our knowledge, this is the first study that quantifies the amount of data movement energy for emerging smart phone applications running on a recent, commercial smart phone device. We hope this characterization study and the insights developed in the paper can inspire innovative designs in smart phone architectures with improved performance and energy efficiency.

I. Introduction

In recent years, there has been an explosive growth in the use of mobile phones, especially smart phones, for our everyday computing needs. To cover the wide range of application requirements running on the smart phones, it is becoming the trend for mobile System-on-Chip (SoC) to include an increasing number of general-purpose application cores (up to eight cores to date), hundreds of graphic processing cores, and many special purpose accelerators such as digital signal processing cores and video encoder/decoder cores, as exemplified by the recent NVIDIA Tegra K1.

Among all the components in a mobile SoC platform, the device display has been shown to be the most energy-hungry in several studies, e.g., [1], [2], when operating in full brightness. In the energy characterization study [2], Pandiyan et al. showed that when the brightness of the screen display is at 25%, the general-purpose application processor becomes the most energy-hungry component, consuming more than 50% of the total energy capacity of the device. This has motivated our work to delve deeper into the energy consumption characteristics of the application cores.

Most prior work that characterize the power profile of smart phones focus on component-level results. For example, Carroll and Heiser [1] used sensors on a smart phone to measure the power usage of the individual computation components under different workloads. Similarly, Murmuria et al. [3] demonstrated a power usage characterization and develop a power-modeling framework, based on the component-level power consumption. However, these component-level energy results do not give the full picture of how energy is spent in the entire system since the information of where data reside in the system and how data is moved across the system are not considered.

Recent work have highlighted the significance of the data movement energy cost in the desktop and server computing environment. Moving data present in the cache requires as much energy as a floating point computation itself and cost much more if the data is not in the cache hierarchy [4]. For scientific applications executed on high-performance desktop/server processors, 28-40% of total energy cost is spent in moving data [5]. The gap between the energy cost of moving data from memory to registers and the energy cost of performing floating point computations is widening 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 [6], [7]. This trend indicates the significance of data movement energy.

To better understand the energy cost of data movement in one of the most energy-hungry elements in smart phones, i.e., the multi-core application processor, this paper performs a detailed investigation to quantify the overall energy cost

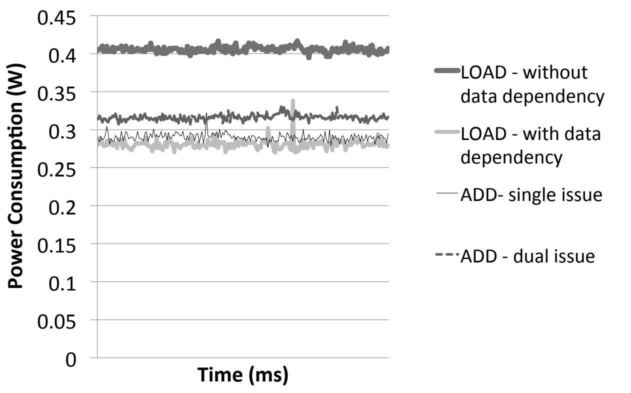

Figure 1. Dynamic power consumption comparison for benchmarks that perform continuous load instruction execution with and without data dependency, and add instruction execution utilizing one or both pipelines of the dual-issue ARM Cortex-A9 processor.

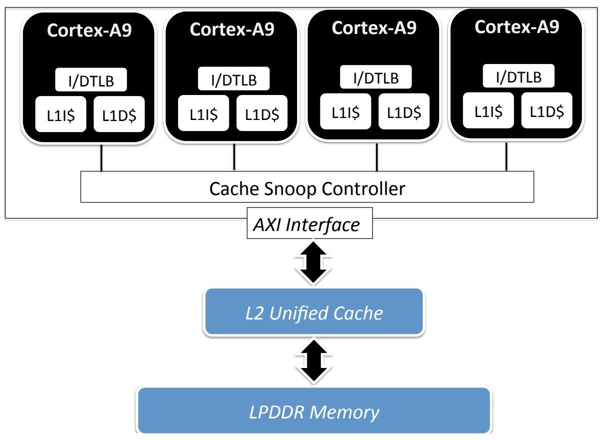

Figure 2. Architecture of the ARM Cortex-A9 processor in a Samsung Galaxy S3 smart phone.

of data movement for modern smart phone applications on a real, commercially available smart phone. We find that even when the mobile application processor is solely working on fetching data from the memory and idling most of the cycles, its power consumption is on par with that when it is busy executing arithmetic operations under 100% utilization.

Figure 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 of the dual-issue ARM Cortex-A9 processor. The dynamic power consumption of the benchmark which performs continuous independent load instruction execution is far more significant than that when performing continuous add operations. This indicates the significance of the data movement energy cost relative to the ALU operations and to the total application energy consumption.

To quantify and evaluate the impact of data movement energy on modern smart phone platforms, this paper first presents a methodology to measure the energy of data movement across the multi-level memory hierarchy in a mobile application processor. We show that the energy cost to perform a memory load instruction whose data is not found in the processor caches is **115** times higher than that of an add operation. With the unit energy cost for the arithmetic and memory operations, we take a step further to characterize modern smart phone workloads running on a Samsung Galaxy S3 platform. 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 interactive smart phone workloads. The data movement energy is particularly high (41%) for realistic web browsing.

In summary, this paper makes the following contributions:

1) We develop a micro-benchmark methodology to characterize in detail the data movement energy across different levels of the memory hierarchy on a commercial smart phone platform. The hardware performance counter- and power meter-based measurements allow us to estimate the energy cost of data movement in the different levels of the memory hierarchy in a Samsung Galaxy S3 device.
2) With the unit cost of data movement energy, we perform a detailed characterization to show the significance of data movement energy and stalled cycle energy of the entire device when running realistic smart phone applications, including various web browsing activities, video playback, photo browsing, and a video game. Based on the result, we offer our insights for future smart phone architecture designs.

The remainder of the paper is organized as follows: In Section II, we give background information of the memory hierarchy for modern smart phone architectures and we describe the design of the micro-benchmarks that characterize the data movement energy for a specific memory hierarchy. Section III outlines our energy measurement methodology. Section IV describes our real-device experimental setup for measuring the energy consumption. Section V presents our experimental results and insights gained from the results. Section VI summarizes prior work in this area and Section VII concludes the paper.

II. Micro-benchmarks: Isolating Data Accesses to a Specific Level of the Memory Hierarchy

A. Background

Modern smart phone processor architectures feature a hierarchical memory structure. Figure 2 illustrates the memory hierarchy of the ARM Cortex-A9 processor in a Samsung Galaxy S3 smart phone. 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

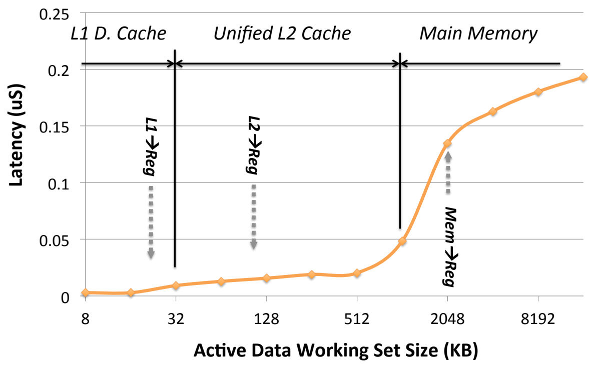

Figure 3. Latency measurement of the designed micro-benchmarks with varying active data working set sizes.

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, we design a set of micro-benchmarks to continuously access data in a specific level of the memory hierarchy. We correlate the data references from the micro-benchmark with power readings obtained from an external power meter to compute the energy cost of each data movement operation.

ARM Cortex-A9 specifications: The designed micro-benchmarks are executed on an ARM Cortex-A9 processor and are used to carry out specific data movement patterns. Since the intended data movement patterns are closely coupled with the architectural parameters of the experimental platform, we give the parameter details here. The ARM Cortex-A9 processor used in this study has four cores. Each core has a private 32KB instruction cache and a private 32KB data cache. All cores share the unified L2 cache whose size is 1MB. In addition, each core also has a two-level translation lookaside buffer (TLB) – L1 instruction and data TLBs (Micro TLBs) are fully set-associative (each with 32 entries) and the L2 TLB (Main TLB) is 2-way set-associative with 128 entries. When virtual-to-physical address translation is not present in the TLB resulting a TLB miss, page table walk needs to be performed. This can cause up to two additional memory accesses in the cache. We use the default page size of 4KB in the experimental platform.

B. Design of the Micro-Benchmarks

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. We design a set of micro-benchmarks that minimize the effect of out-of-order execution and other architectural optimizations. The design of the micro-benchmark methodology is inspired by a recent work that quantifies the data movement energy cost for scientific applications running on desktop and server processors [5].

```
Initialize benchmark
Start Timer
for $i = 0$ TO $i < iterations/x$ do
    Unroll $x$ times { benchmark operation; }
end for
Stop Timer
```

Algorithm 1: Pseudo-code of the micro-benchmark which performs the desired data movement in a target level of the memory hierarchy iteratively.

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 micro-architectural 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. We summarize the workings of our micro-benchmarks as follows:

1) Each micro-benchmark first requests the operating system to allocate a size of memory 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 to avoid overhead due to array index increment. We utilize a temporary pointer variable to hold an address that refers to a word in the data set and a subsequent dereference of the 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;
   ```
3) By continuously performing these dereferences in a loop, we traverse different portions of 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 micro-benchmark indeed performs the intended memory accesses in the target level of the memory hierarchy. Algorithm 1 outlines the pseudo-code of the micro-benchmark.

We create three micro-benchmarks to study the data movement energy across the different levels of the memory hierarchy, namely from the L1 cache to registers, from the

L2 cache to registers, and from main memory to registers.

- **L1 cache to CPU register** ($Microbenchmark_{L1 \rightarrow Reg}$): This micro-benchmark 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. We choose a data set of 24KB which comfortably fits in the 32KB L1 data cache.
- **L2 cache to CPU register** ($Microbenchmark_{L2 \rightarrow Reg}$): This micro-benchmark 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 micro-benchmark 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, we need to choose a data set size that fits in the 1MB L2 cache but does not fit in the 32KB L1 cache. In addition, we also want to keep the TLB miss rate as small as possible[1]. To fulfill these constraints, we select the data set size of $Microbenchmark_{L2 \rightarrow Reg}$ to be 125KB. As shown in Figure 3, 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** ($Microbenchmark_{memory \rightarrow Reg}$): This micro-benchmark 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 micro-benchmark inevitably thrashes the L2 TLB. Because the additional memory references related to page table walks also access the cache hierarchy, these additional references can significantly lower the high L1 and L2 cache miss rates expected for the random access micro-benchmark. 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, we experimentally choose a data set size that has a high L2 miss rate while keeping the TLB allocations as few as possible. In this paper, we choose the data set size to be 2MB. Therefore, this causes more data movement for each memory instruction compared to one with a smaller data set.
- **Integer and NOP** ($Microbenchmark_{add}$ **and** $Microbenchmark_{nop}$): In addition to the micro-benchmarks which perform iterative data movement from a specific level of the memory hierarchy to the processor register, we design two more micro-benchmarks that execute integer addition and NOP instructions continuously to understand their relative impact on energy consumption with respect to that of data movement.

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

C. Discussion

There are other challenges and considerations in the process of designing and running the micro-benchmarks, which we discuss below.

- **Compiler Optimization:** It is possible for the compiler to optimize away the register variable allocated for the temporary pointer storage. Because of this, we manually verify the designed data movement patterns by looking at the assembly code generated for the micro-benchmarks.
- **Loop Unrolling Effect:** The main loop of memory accesses in the micro-benchmarks can be unrolled such that a larger number of memory loads are executed for each loop iteration to reduce the frequency of branch instruction execution. We carefully select the loop size/iterations 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.
- **Priority of the Micro-benchmarks, Task Migration, and Dynamic Frequency Control in the Operating System:** To minimize interference with other running processes, we give the micro-benchmarks the highest priority by setting the `nice` number[2] to -20. Furthermore, to prevent task migration between the different, available cores, we pin the micro-benchmarks to a specific core at the beginning of the program execution. Finally, the micro-benchmark power consumption highly depends on the core frequency setting, and the default Android/Linux operating system varies the core frequency dynamically based on utilization. To eliminate the influence of frequency variation, we use the *performance* governor to control and set the frequency of the cores at 1.4 GHz.

III. Data Movement Energy Measurement

In this section, we describe the techniques we used to measure the energy cost of individual instructions and to isolate the components of the system that do not contribute to the data movement energy, e.g., idle energy and stalled cycles. Then we derive the energy cost of data movement and cache prefetchers.

[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.

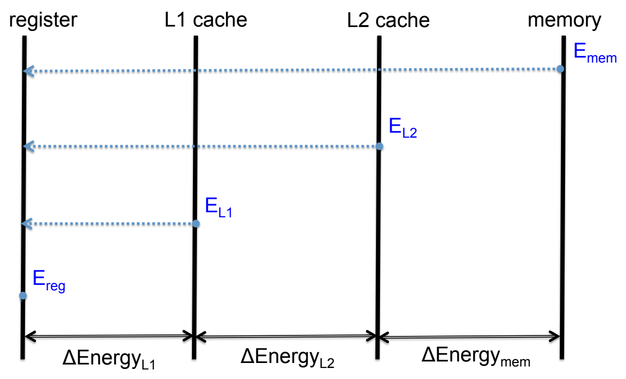

Figure 4. Energy measurement for data movement across the memory hierarchy.

Figure 4 illustrates our approach in determining the energy cost for accessing data in different levels of the memory hierarchy. We first determine the energy cost for moving data from the L1 cache to the registers ($\Delta Energy_{L1}$), and repeat this process 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 micro-benchmarks is in the order of billions, we ensure that the majority of the processor energy consumption is spent on the designed data movement.

A. Dynamic Energy Measurement of the Micro-Benchmarks

Working with a commercial product does not offer the flexibility of a product development board with exposed test points/voltage rails or even that of a PC motherboard. The only access points our system offers for power measurement are the battery terminals. The entire system that consists 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, we keep all the peripheral components turned off or 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 [2] since the micro-benchmarks used in our measurements do not make use any peripheral components. To minimize potential interference, we also manually terminate unnecessary background service processes in our smart phone experimental platform that can potentially skew measurements of the micro-benchmarks. The power measurements are made in stable memory access phases (regions of interest) when the program is fully loaded into memory and executes only data movement instructions.

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

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

Furthermore, the total energy consumption of the micro-benchmarks is

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

B. 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 spends 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 micro-benchmarks.

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, we design a matching micro-benchmark, $Microbenchmark_{L1\rightarrow Reg,no-dep.}$, which performs exactly the same data movement as in $Microbenchmark_{L1\rightarrow Reg}$, except that all data dependencies in the original micro-benchmark have been removed. This is done by replacing the pointer chasing-based data movement with pre-calculated memory addresses. 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:

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

C. 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. We use the same formula below for these long latency operations.

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

D. 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 $E_{RAM \to L2}$, 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 [5]. Isolating the energy consumption of prefetch engine's overhead from this is not apparent, due to a combination of the energy measurement methodology we adopt 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 [8].

The per-core L1 cache prefetchers monitor cache references to the L1 cache based on the program counter (PC) value and address and is 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.

IV. Experimental Setup

This section introduces our experimental methodology for real-system energy measurements.

A. Real-Device Experimental Platform

We perform our experiments on a Samsung Galaxy S3 smart phone which houses a Samsung-made Exynos4 Quad 4412 SoC. The SoC has four Cortex-A9 application cores with a 1GB Low Power DDR (LPDDR) memory. There are separate L1 instruction and data caches for each of the four cores and a shared unified L2 cache. The device runs a rooted Android 4.3 Jelly Bean OS. The Linux kernel is configured to enable performance profiling with ARM Streamline [9]. The micro-benchmarks are written in C, cross-compiled on the host machine with ARM-Android NDK toolchains [10]. The binaries are pushed to the device and are launched from the host machine via the adb terminal.

Table I
Energy cost of data movement.

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

Cortex-A9 provides 6 hardware performance counters which can be used to measure six performance events simultaneously. Similarly, the L2 cache controller that is integrated with the application cores provides two counters. The L2 prefetcher is turned on/off by modifying a part of architecture-specific kernel code that runs during the OS initialization. We extend the Linux kernel modules in [11] to read the L2 cache controller registers and to validate the configuration that has been set by the kernel.

B. Energy Measurement for the Experimental Platform

We remove the smart phone's battery and power the device with a DC power supply set to 4.0V. The National Instruments DAQ 6251 [12] is used for voltage and current measurement along with a small shunt resistor. The DAQ periodically samples voltages which is graphically represented in the NI Signal Express [13] tool. 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 energy measurement results presented in this paper are obtained with the lowest brightness setting for the display. And, the results are the average of three individual runs of each experiment.

C. Workloads

In addition to the set of micro-benchmarks used to quantify the energy cost of data movement, we execute a set of commonly-used smart phone workloads for data movement energy characterization. We use smart phone applications from the MobileBench suite [2], including an image processing application (`PhotoView`), video playback (`VideoPlayback`), general web browsing (`GWB`), realistic web browsing (`RWB`), and education-oriented web browsing (`EWB`).

`PhotoView` is a photo-browsing workload that starts a slide show of full-screen high resolution images, each with

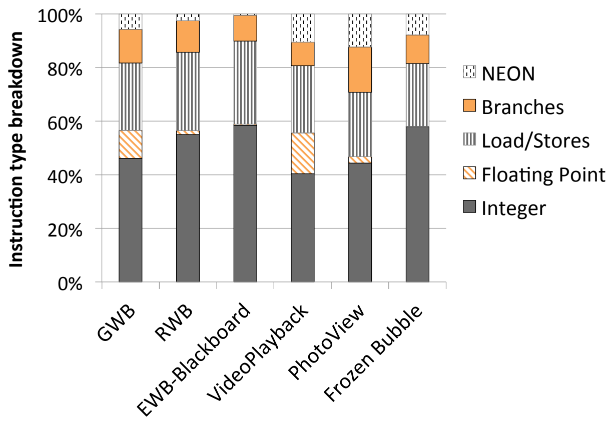

Figure 5. Instruction mix for workloads

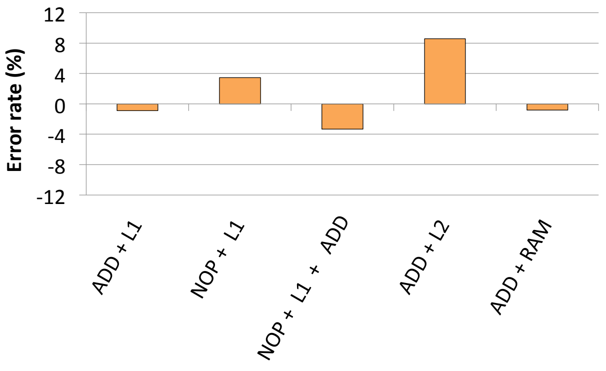

Figure 6. Validation error rate.

a resolution of 4912x3264 and is of size between 4 to 6 MB, whereas `VideoPlayback` renders a full-screen HD (720p) MPEG-4 video of size 80 MB. `GWB` and `RWB` sequentially load and render a set of popular websites, including Amazon, BBC, CNN, Craiglist, eBay, ESPN, Google, MSN, Slashdot, Twitter, and YouTube, from the secondary storage. The benchmarks differ in the browsing activities performed after the web pages are fully loaded. `RWB` mimics realistic user browsing behavior by scrolling the web pages up and down with random delays and web page zooming. `EWB` is based on the popular Blackboard course web platform. It combines Blackboard page browsing and document browsing by invoking a document reader to skim through a set of Portable Document Files (PDFs). Furthermore, we evaluate a gaming workload, Frozen Bubbles, which models an interactive game application [14].

Figure 5 shows the instruction composition for these workloads. About 60% of the instructions (i.e., integer, floating point and NEON instructions) perform computations and a significant 25% of the instructions perform memory loads and stores. With the energy consumption of a single instruction or data memory access[3] potentially costing upto 115 times of the cost of an integer instruction, data movement energy cost evaluation for mobile workloads is critical. We present a detailed analysis of the data movement energy cost in Section V.

D. Validation

We create another set of benchmarks to validate our energy estimation methodology and the measured energy cost values. Each of these validation benchmarks essentially performs a combination of the data movement and arithmetic operations described earlier in Section II. Performance counter values are collected for the benchmarks from Streamline to identify a) the number of data movement operations that occur between the different levels of the memory hierarchy, b) the number of stall cycles, and c) the number of arithmetic operations.

The energy consumed by each of these operations over the benchmark execution is calculated by multiplying the energy cost per unit operation (Table I) with the raw counts given by the performance counters. For example, the total energy consumption of the $L2 \rightarrow Reg + Add$ validation benchmark is

$$\begin{aligned} E_{L2\rightarrow Reg+Add} = E_{L2\rightarrow L1} * N_{L2} + E_{L1\rightarrow Reg} * N_{L1} \\ + E_{Stall} * N_{stall} + E_{Add} * N_{Add} \end{aligned}$$

The estimated energy consumption is compared with the actual energy consumption measurement of the experimental platform. Figure 6 shows the accuracy of our micro-benchmark methodology for data movement energy. The error rate is calculated as follows:

$$Accuracy = \frac{E_{measured} - E_{estimated}}{E_{measured}}$$

We find that our micro-benchmark approach can accurately estimate the energy consumption of data movement. The error rate is 3.4% on average and is generally under 3.5%. Only for the $L2 \rightarrow Reg+Add$ benchmark, we over-estimate the benchmark energy consumption by 8.6%.

V. Energy Cost Analysis for Mobile Workloads

We obtain the total energy consumption measurement for a diverse set of mobile workloads (Section IV-C) by sampling the dynamic power consumption, P_i, of a mobile application with the DAQ continuously. The total energy consumption of the application is derived from the dynamic power consumption samples as follows:

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

The total energy consumption includes the dynamic energy consumption of the entire experimental device but not

[3]For example, the execution of a memory load instruction needs to access the memory twice – the instruction itself and data in the requested memory location.

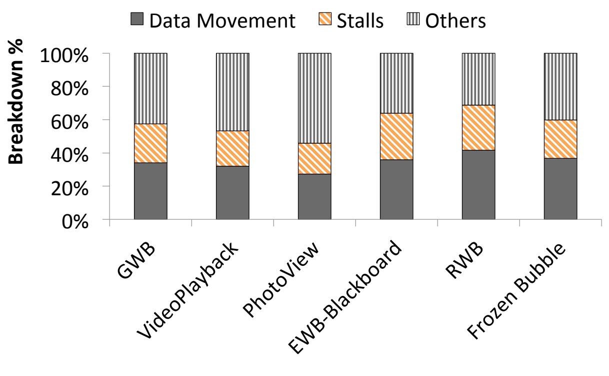

Figure 7. Energy breakdown for the experimental device.

considering the idle energy portion. 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 cost for moving data between the different levels (Table I). Similarly, we evaluate the energy spent on processor stall cycles. 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 shows the energy breakdown for the mobile workloads. The *Data Movement Energy* bars represent the portion of the total device energy consumption attributed to the data movement in the application processor's memory hierarchy and the *Stall Cycle Energy* bars represent the portion of the total device energy consumption attributed to processor stall cycles while 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 behavior of using hardware acceleration for *jpeg* decoding. As a result, more energy is spent on the *Others* category for `PhotoView`.

What we also observe 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 when we consider realistic user behavior for mobile devices. Users typically do not use their smart phones for continuous computations. Typical smart phone 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 amount 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%. We expect a significant portion of the $Other$ energy consumption to come from the smart phone display, which has been shown as one of the most power-hungry components in modern smart phones [1], [2].

In addition to the energy breakdown, we also perform the energy analysis for data moving from one level of the memory hierarchy to another level. Figure 8 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 on the memory access patterns of the mobile workloads, the energy consumption dedicated to each level of the memory hierarhcy 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 our 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. [15]. 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.

VI. Related Work

There have been a number of efforts that develop power models to estimate smart phone energy consumption. Most

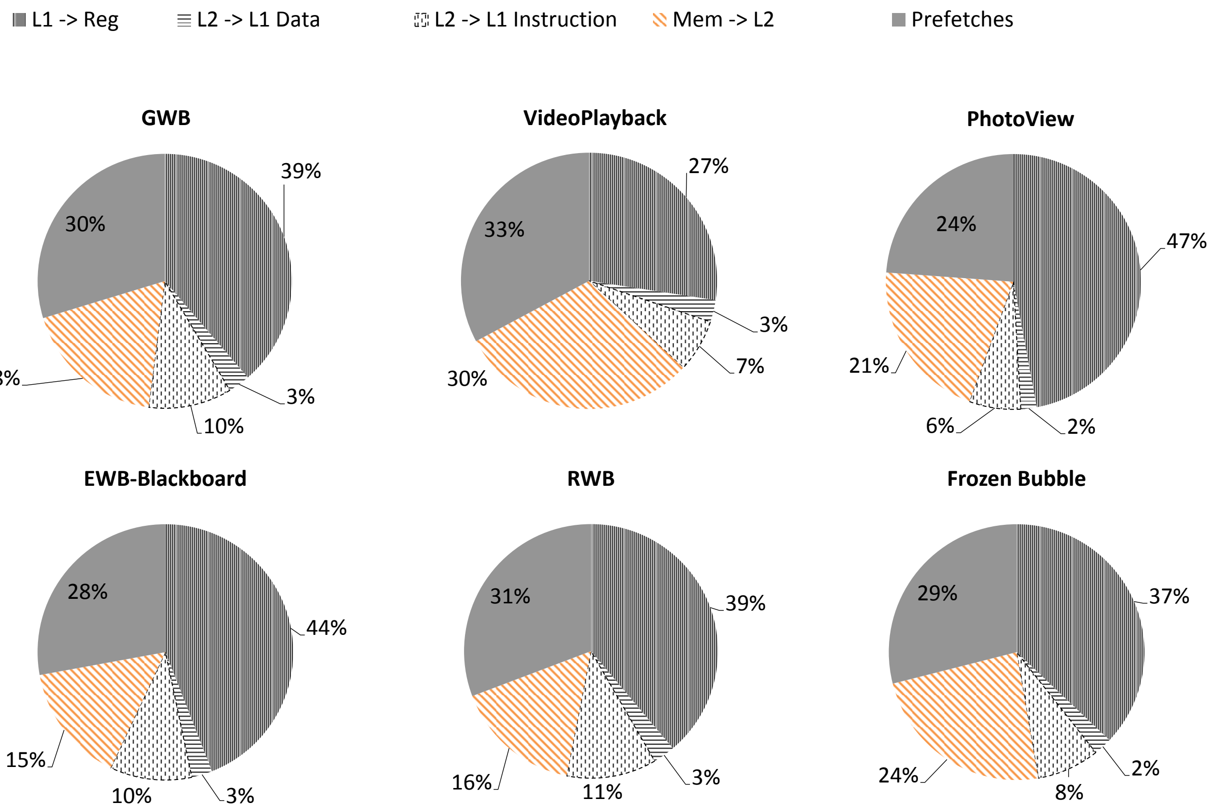

Figure 8. Relative energy cost for moving data from one level to another level of the entire memory hierarchy in the ARM Cortex-A9 processors for mobile workloads.

of these power modeling and energy estimation techniques are based on application processor utilization. Shye et al. [16] described a regression-based power estimation technique that can predict the power consumed by different components on a smart phone. Pathak et al. [17] developed a power model based on system call tracing and demonstrated that energy estimation is possible using their developed power model. Zhang et al. [18] leveraged the battery sensor on a smart phone to construct a model for power estimation. Li et al. [19] took a different approach to energy estimation by profiling the Java byte code of an application, aiming at providing energy consumption data to application developers. However, none of these works focus on data movement energy consumption which is a major contributor to the overall device energy consumption, as we show in this paper.

There have been a few studies that quantify data movement energy via micro-benchmarking for server processors. Molka et al. [20] quantified instruction level and data transfer energy consumption for Intel and AMD server processors. Kastor et al. [5] took a step further by proposing an energy cost evaluation methodology for data movement between the different memory levels. With the estimated data movement energy cost, the data movement energy characterization for scientific applications were performed. Nonetheless, both studies were performed on server-class processors. This work is the first that attempts to quantify the data movement energy cost for modern smart phone workloads running on a commercial smart phone device. The smart phone applications and the mobile processors are vastly different from their server or desktop counterparts. With the increasing popularity of smart phones, the data movement energy cost presented in this paper along with the detailed data movement energy characterization for the diverse range of smart phone applications provide insights into which data paths to improve upon, in order to further increase the energy efficiency of next generation smart phones.

VII. Conclusion

This paper presents a detailed methodology for quantifying the data movement energy cost on a commercial smart phone and summarizes the energy cost for moving data from one to another level of the memory hierarchy in a mobile processor, ARM Cortex-A9. Given the energy cost, we take a step further to characterize the portion of the energy that is spent on data movement for a diverse set of smart phone applications. Overall, the energy spent on data movement in mobile processors is significant. 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 interactive smart phone workloads. The data movement energy is particularly high (41%) for realistic web browsing that is commonly performed on smart phones. Our results also indicate a relatively high stalled cycle energy consumption (an average of 23.5%) for current smart phones. This motivates mobile processors to include more but simpler cores in the design. Furthermore, the considerable amount of energy spent on moving data from the memory to the L2 cache encourages more researches into low-power emerging memory technologies for embedded devices. With the detailed energy characterization and the insights provided in this paper, we hope to inspire innovative designs that lower power consumption of memory instructions through more optimized data paths and simpler architecture for smart phone architectures.

Acknowledgment

The authors would like to thank the anonymous reviewers for their feedback. This work was partially supported by the NSF I/UCRC Center for Embedded Systems (NSF grant #0856090) and by Science Foundation of Arizona under the Bisgrove Early Career Scholarship. The opinions, findings and conclusions or recommendations expressed in this manuscript are those of the authors and do not necessarily reflect the views of the Science Foundation of Arizona.

References

[1] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of USENIX annual technical conference (USENIX-ATC)*, 2010.

[2] D. Pandiyan, S.-Y. Lee, and C.-J. Wu, "Performance, energy characterization and architectural implications of an emerging mobile platform benchmark suite – MobileBench," in *Proceedings of International Symposium on Workload Characterization*, 2013.

[3] R. Murmuria, J. Medsger, A. Stavrou, and J. Voas, "An analysis of power consumption in a smartphone," in *Proceedings of International Conference on Software Security and Reliability*, 2012.

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

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

[6] 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.

[7] B. Dally, "Power, programmability, and granularity: The challenges of ExaScale computing," in *Proceedings of International Parallel and Distributed Processing Symposium*, 2011.

[8] "PL310 Cache Controller Technical Reference Manual," http://goo.gl/GUP7xf.

[9] "ARM STREAMLINE ANALYZER," http://ds.arm.com/ds-5/optimize/.

[10] "Android NDK," https://developer.android.com/tools/sdk/ndk/index.html.

[11] "Cortex A9 prefetch disable," https://github.com/deater/uarch-configure/tree/master/cortex-a9-prefetch.

[12] "National Instrumens PXI-6251 DAQ," http://sine.ni.com/nips/cds/view/p/lang/en/nid/14125.

[13] "NI SignalExpress," http://www.ni.com/labview/signalexpress.

[14] Y. Huang, Z. Zha, M. Chen, and L. Zhang, "Moby: A mobile benchmark suite for architectural simulators," in *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, 2014.

[15] A. Gutierrez, R. Dreslinski, T. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *Proceedings of International Symposium on Workload Characterization*, 2011.

[16] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: Studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the International Symposium on Microarchitecture*, 2009.

[17] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of European Conference on Computer Systems*, 2011.

[18] L. Zhang, B. Tiwana, R. Dick, Z. Qian, Z. Mao, Z. Wang, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis*, 2010.

[19] D. Li, S. Hao, W. G. J. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the International Symposium on Software Testing and Analysis*, 2013.

[20] D. Molka, D. Hackenberg, R. Schone, and M. S. Muller, "Characterizing the energy consumption of data transfers and arithmetic operations on x86-64 processors," in *Proceedings of International Conference on Green Computing*, 2010.