Understanding the Sources of Power Consumption in Mobile SoCs

Mostafa Said[1], Sofiane Chetoui[1], Adel Belouchrani[2], Sherief Reda[1]
[1]SCALE Lab, Brown University, Rhode Island, USA
[2]Ecole National Polytechnique, Algeria
Emails: {mostafa_said,sofiane_chetoui,sherief_reda}@brown.edu, adel.belouchrani@enp.edu.dz

***Abstract*—In this paper we propose a fine-grain blind leakage and dynamic power identification technique, and we use it to analyze the dynamic and leakage power of mobile SoCs at the core level. We also introduce a new experimental methodology to apply blind power identification for heterogeneous SoCs, including a novel initialization for the algorithm that enhances its power estimation accuracy. We shed light on power usage using real life applications, showing how power is divided among the big, the LITTLE cores and the GPU. Our results show that for some applications, the GPU can have the highest power consumption, and that the LITTLE cluster can have large values of leakage power. We also elucidate the trade-offs between power consumption and performance of the big cluster versus the little cluster of the SoC at different frequencies. We also show how the power is affected by CPU and skin thermal throttling.**

***Index Terms*—Multicore Processors; GPU; Leakage Power**

I. Introduction

Researchers have investigated a wide range of methods to characterize power consumption in mobile platforms [1]–[5]. For instance, Wang *et al.* [1] and Zhang *et al.* [2] analyzed the power consumption of each component in a smartphone, where the CPU power is estimated as one unit without further insight about the power consumption of each core inside the CPU. A number of techniques were proposed to analyze the dynamic and leakage fractions in the power consumption of components [3]–[5]. However, these analyses were conducted using assumptions that are not always true. For example, sometimes a constant ratio between dynamic and leakage power is assumed [3], [4], which is not always correct, because the ratio between leakage and dynamic power changes with the operating frequency, temperature, supply voltage, and the activity factor. Also it was assumed that all cores have the same design and area [5]; however, modern SoCs use big.LITTLE heterogeneous architectures.

In this paper we analyze the dynamic and leakage power at the core level by proposing a novel fine-grain power modeling approach, which is valid for both heterogeneous and homogeneous systems. We observe that the per-core power given by the Blind Power Identification (BPI) [6], [7] are equal to leakage power whenever the corresponding core is idle. Relying on this observation and using regression analysis, we build a leakage power model per core. Finally, the leakage power models are used along with the power components obtained by the BPI algorithm to get the per-core dynamic and leakage power values.

The analysis gives for the first time an insight about the dynamic and leakage power values at a fine-grain level of a heterogeneous mobile SoC. Our method is attractive as it does not need prior knowledge of the physical layout or any other design parameters. Second, it only relies on fine-grain temperatures and total power consumption, and does not need any additional measurements (e.g., performance counters or the operating frequencies) [2], [8], [9]. Another contribution of this paper is enhancing the accuracy of the BPI, where we introduce a novel initialization setup that reflects the physical relationship between temperature and power consumption, which reduces the power estimation error from 5.04% to 3.77% compared to [7].

We implement our method on an Android-based smartphone and use it to analyze the dynamic and leakage power consumption of its SoC. We elucidate the power consumption of the big cores, little cores and GPU using various real-life workloads. Our results show that for some applications, the GPU can have the highest power consumption, and that the LITTLE cluster can have large values of leakage power. Furthermore, we provide the performance-power trade-offs of using the big cluster versus the little cluster, and describe the conditions where the little cluster can provide better performance and energy consumption. Furthermore, we demonstrate the various thermal conditions that can lead to throttling for the SoC, leading to loss in performance.

This paper is organized as follows. In Section II we provide the essential background for BPI. In Section III we describe our power identification methodology to apply BPI for heterogeneous SoCs in mobile platforms to identify the leakage power model for each unit. In Section IV we provide a comprehensive power analysis results using a simulation tool chain and a real mobile SoC. Finally, Section V summarizes the main conclusions of the paper.

II. Background

In a discretized form, the physical relationship between temperature and power is given by [9], [10]:

$$\mathbf{t}(k) = \mathbf{A}\mathbf{t}(k-1) + \mathbf{B}\mathbf{p}(k) + \epsilon(k), \tag{1}$$

where, $\mathbf{t}(k)$ and $\mathbf{p}(k)$ are $N \times 1$, vectors that denote the temperature and power consumption measurements of the N cores at time k, respectively; $\mathbf{A}$ and $\mathbf{B}$ are the two modeling

978-1-5386-7466-6/18/$31.00

matrices that capture the physical relationship between power and thermal; and $\epsilon(k)$ is a vector that represents the noise in the measurement process at time k. The blind estimation problem seeks to identify the matrices $\mathbf{A}$ and $\mathbf{B}$, together with the power profiles $\mathbf{p}(k)$.

It is well known that the steady-state thermal model can be derived from Equation 1. If a stable set of power sources, denoted by the vector $\mathbf{p_s}$, are applied, then after the transient response, the steady-state (SS) temperatures, denoted by the vector, will be measured; i.e., $\mathbf{t_s} = \mathbf{t}(k \rightarrow \infty)$. By re-arranging Equation 1, one gets

$$\begin{aligned} \mathbf{t_s} &\approx \mathbf{A}\mathbf{t_s} + \mathbf{B}\mathbf{p_s}, & (2)\\ \mathbf{t_s} &\approx (\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}\mathbf{p_s}, \\ \mathbf{t_s} &\approx \mathbf{R}\mathbf{p_s}, & (3) \end{aligned}$$

where, $\mathbf{R} = (\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}$ is the *steady–state thermal transfer matrix* and $\mathbf{I}$ the identity matrix. If one obtains M thermal steady-state measurements, $[\mathbf{t_{s1}}\ \mathbf{t_{s2}}\ \ldots\ \mathbf{t_{sM}}]$, from different experiments using M different sets of power signals, $[\mathbf{p_{s1}}\ \mathbf{p_{s2}}\ \ldots\ \mathbf{p_{sM}}]$, then we can summarize the results using

$$\begin{aligned} [\mathbf{t_{s1}}\ \mathbf{t_{s2}}\ \ldots\ \mathbf{t_{sM}}] &= \mathbf{R}[\mathbf{p_{s1}}\ \mathbf{p_{s2}}\ \ldots\ \mathbf{p_{sM}}] & (4)\\ \mathbf{T_s} &= \mathbf{R}\mathbf{P_s}. & (5) \end{aligned}$$

If $\mathbf{R}$ and $\mathbf{A}$ are identified, matrix $\mathbf{B}$ can be calculated by $\mathbf{B} = (\mathbf{I}-\mathbf{A})\mathbf{R}$. In practical cases, the thermal sensors can provide the temperature of each SoC unit, while usually only one power sensor is provided at the SoC level. Thus, only the left-hand side of Eq. 5 is known while the sum of each column in $\mathbf{P_s}$, i.e. total power, is known in the right-hand side. The problem is then how to identify $\mathbf{R}$ given only the measurements of the thermal sensors and total power. BPI solves this problem in three steps [7]:

i. The Non-negative Matrix Factorization (NMF) algorithm is first used to factor the given temperature matrix $\mathbf{T_s}$ into two non-negative matrices: a $N \times N$ matrix $\tilde{\mathbf{R}}$ and a $N \times M$ matrix $\tilde{\mathbf{P_s}}$, such that $\mathbf{T_s} = \tilde{\mathbf{R}}\tilde{\mathbf{P_s}}$.

ii. The estimations of $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{P_s}}$ must be further processed to conform with the physical properties that both should have in a realistic system. First, the maximum element of each column in $\tilde{\mathbf{R}}$ matrix should lie on the diagonal to match the fact that a power source in some core should have maximum thermal impact on its core and smaller impact on other cores because of heat diffusion. This requires a reordering of the columns of the $\tilde{\mathbf{R}}$ matrix and the corresponding rows in $\tilde{\mathbf{P_s}}$ such that maximum element in each column of $\tilde{\mathbf{R}_s}$ ends up positioned on the diagonal in the new column ordering. To avoid introducing new symbols, we refer to the ordered matrices as $\tilde{\mathbf{R}_s}$ and $\tilde{\mathbf{P_s}}$.

iii. There is no guarantee that the sum of each column in $\tilde{\mathbf{P_s}}$ is equal to the total measured power that each column represents. Therefore scaling factors, $\alpha_1, \ldots \alpha_N$ are introduced such that:

$$\begin{aligned} [\alpha_1\ \alpha_2\ \cdots\ \alpha_N]\,\tilde{\mathbf{P_s}} &= \mathbf{p_{tot}}, & (6)\\ \mathbf{p_{tot}} &= [p_{tot-1}\ p_{tot-2}\ \cdots\ p_{tot-M}] & (7) \end{aligned}$$

where $\mathbf{p_{tot}}$ is a row vector that represents the total power measurements for each vector in $\tilde{\mathbf{P_s}}$, where p_{tot-i} is the total power measured that is equal to the summation of the elements of column i in $\tilde{\mathbf{P_s}}$. After solving the system of Eq. 6 for the α's, we can update the values of $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{P_s}}$ to get the final required $\mathbf{R}$ and $\mathbf{P_s}$ as follows:

$$\mathbf{R} = \begin{bmatrix} 1/\alpha_1 & 0 & \cdots & 0 \\ 0 & 1/\alpha_2 & \cdots & 0 \\ \vdots & & \ddots & 0 \\ 0 & 0 & \cdots & 1/\alpha_N \end{bmatrix} \tilde{\mathbf{R}} \quad (8)$$

$$\mathbf{P_s} = [\alpha_1\ \alpha_2\ \cdots\ \alpha_N]\ \tilde{\mathbf{P}} \quad (9)$$

During runtime, a quadratic programming algorithm is used to determine the power of individual system units using the following optimization setup:

$$\begin{aligned} \min \quad & \|\mathbf{B}\mathbf{p}(\mathbf{k}) - \left(\mathbf{t_{meas}}(\mathbf{k}) - \mathbf{A}\mathbf{t_{meas}}(k-1)\right)\|_2^2 \\ \text{s.t.} \quad & \mathbf{p}(k) \succeq 0 \\ & \|\mathbf{p}(k)\|_1 = \mathbf{p_{meas-tot}}(k) \end{aligned} \quad (10)$$

where $\mathbf{t_{meas}}$ is the thermal sensors measurements vector per individual system units and $\mathbf{p_{meas-tot}}$ is the total power measured for the set $\mathbf{t_{meas}}$. The solution of that system is $\mathbf{p}(k)$, which is the desired power consumption of every unit in the SoC at time k.

III. The Proposed Methodology for Leakage Identification

We first propose in Subsection III-A a novel approach that allows us to break down the power estimate from the BPI algorithm to characterize the leakage power at the core level. Furthermore, since our method relies partially on the BPI, whose output is sensitive to the initialization, we propose a new initialization methodology in Subsection III-B that decreases the error of the BPI output.

A. Leakage Power Modeling

In this section a fine-grain approach is proposed to determine the leakage and dynamic power of individual cores. We observe that the BPI power estimate for a core corresponds to the leakage power of that core whenever the core is in the idle mode. Using these leakage power values along with their corresponding temperature values, we estimate the different parameters of the leakage power model. Finally, the leakage power models are used along with the BPI to identify both the leakage and dynamic power at the core level.

The approach is composed of an offline and online steps. In the offline step the parameters of the leakage models per core are estimated as illustrated in Fig. 1. The first step is to stress the different cores while keeping the others in the idle mode.

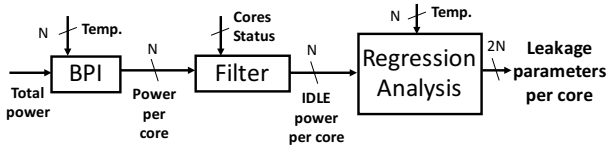

Fig. 1: Offline steps of leakage power parameters estimation.

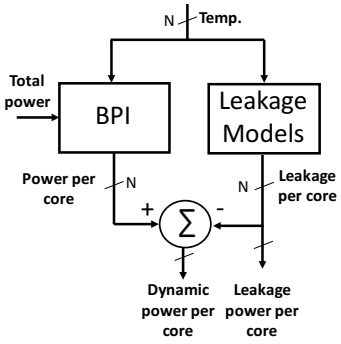

Fig. 2: Online identification of leakage and dynamic power.

Throughout the experiments we keep track of the total power consumption, the temperature measurements of the different cores and their corresponding online/offline state. Afterwards, the temperature values and the total power consumption are given as inputs to the BPI algorithm, which then outputs the power consumption per core. Next, we filter the power values based on the status of the cores such that we only keep the power corresponding to idle mode. Finally, the idle power per core is used along with the corresponding temperature values to estimate the leakage power parameters using a non linear least square regression analysis based on the following leakage power model:

$$P_{i_{leakage}}(k) = c_{i_1} V_{dd} t_{i_k}^2 e^{\frac{-c_{i_2}}{t_{i_k}}}, \tag{11}$$

where c_{i_1} and c_{i_2} are constants unique to core i, V_{dd} is the supply voltage and t_{i_k} is the temperature at time k of core i. The exponential power model was chosen after trying several models, including linear ones, the choice was based on least square error between the model and the real power values. Based on this mathematical model, and the presence of a wide range of temperature values along with their corresponding single core power consumptions, the two parameters c_{i_1} and c_{i_2} are estimated for each core.

The estimated parameters are used afterwards in the online step to estimate the leakage power based on the temperature values as shown in Fig. 2. Meanwhile the temperature values along with the total power consumption are given as inputs to the BPI algorithm to get the total power consumption per core. Finally, the per-core leakage power values are subtracted from the total power of the corresponding cores to get the dynamic power.

B. BPI Initialization for Heterogeneous Systems

We use the same NMF algorithm used in [6], [7] which is based on block-coordinate update [11]. The output of this algorithm is dependent on the initial values of both $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{P}}_\mathbf{s}$. It was reported in [7] that the quality of the solution is very sensitive to the initialization, where they proposed to initialize $\tilde{\mathbf{R}}$ to the identity matrix since it is a symmetric matrix and the diagonal elements are the maximum elements in each column. We observe that the initialization should reflect the physical characteristics of both matrices. Therefore, to reflect more the underlying system and the characteristics of both $\tilde{\mathbf{R}}$ and $\tilde{\mathbf{P}}_\mathbf{s}$, we propose the following initialization procedures:

Non diagonal elements of $\tilde{\mathbf{R}}$: Any two symmetrical elements $\tilde{\mathbf{R}}_{ij}$ and $\tilde{\mathbf{R}}_{ji}$, where $i \neq j$ are initialized to the average of their steady-state temperature values when their corresponding cores, i.e., core i and core j, are stressed.

Diagonal elements of $\tilde{\mathbf{R}}$: We first observe that cores that show similar thermal characteristics should have the same values in their diagonal elements. We define two cores to have *similar* thermal characteristics if they show the same steady-state temperature, within 1% of difference, under the same stress kernel and the same conditions, i.e. frequency, ambient temperature. We reflect this similarity in the diagonal elements corresponding to similar cores by initializing their diagonal elements to be equal to the average of their steady-state temperatures. When there is no similarity detected for a core, its corresponding diagonal element in $\tilde{\mathbf{R}}$ is simply initialized to the steady-state temperature of its core, when its core is the only core stressed. For the GPU, it was inevitable to stress it while at least one of the cores is online. Therefore we used the same initialization used in [7] for the corresponding row and column of the GPU.

Initializing $\tilde{\mathbf{P}}_\mathbf{s}$: The initialization used in [7] simply divides the total power for each stress pattern equally among the cores. In our study we depend on the direct proportionality between steady-state temperature and power so we divide the total power for each stress pattern according to the ratio between the core temperature and the summation of all temperatures. For example, assuming for stress pattern j, the total measured power is p_{tot-j} then the entry in row i and column j of $\tilde{\mathbf{P}}_\mathbf{s}$ will be initialized to $p_{tot-j} \times t_i / (\sum_j {}^{N}_{j=1} t_j)$ where t_i is SS temperature of core i.

IV. Experimental Results

First we verify the accuracy of the proposed leakage identification methodology and the proposed BPI initialization in HotSpot thermal simulator. Then the proposed methodology is applied and experimented on an Android-based Google Pixel 2 XL mobile platform.

A. Verification Using HotSpot

We verify both the proposed leakage identification method and our new initialization approach using the HotSpot thermal simulator [12], where we can control the input power traces and compare BPI results against the known inputs.

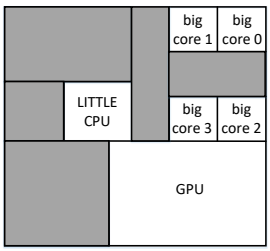

Fig. 3: Layout of the heterogeneous SoC presented in [13].

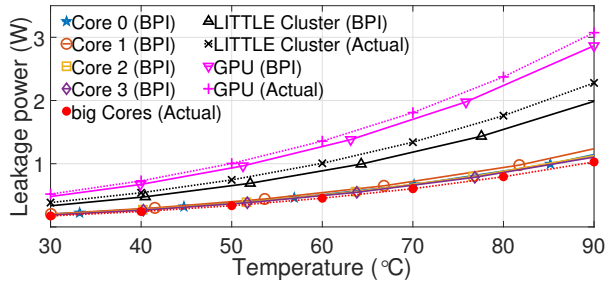

Fig. 4: The BPI leakage power estimations of the four big cores and both the GPU and LITTLE cluster in comparison to the actual synthetic models of HotSpot (dotted curves).

1) Leakage identification verification in HotSpot: The HotSpot tool [14] supports leakage power modeling such that the output temperature map is computed based on the user input power and the SS temperature in a positive feedback loop. This feedback loop continues until the increase in SS temperature becomes minimal and smaller than some predefined constant. We replace the default leakage model inside HotSpot with the one in Eq. 11, where we tune the parameters c_1 and c_2 to avoid thermal runaway at higher temperatures and to provide realistic leakage values. We choose the benchmark floorplan of [13] because it represents a big.LITTLE+GPU system. This layout is shown in Fig. 3 where the shaded areas are for other units that we are not targeting in this study. In that simulation we apply our proposed methodology in Section III to identify the leakage power of the 4 big cores (core0, core1, core2, core3) and both the GPU and the LITTLE cluster[1]. The estimated models are shown in Fig. 4. As shown, the GPU has the highest leakage due to its large area. The LITTLE cluster has higher leakage than any of the big cores, but the big cluster combined shows much higher leakage than the LITTLE cluster. To measure the accuracy of our methodology, we calculate the average percentage error (Avg. % err.) as

$$Avg.\ \%\ err. = \frac{1}{n}\sum_{i=1}^{n}\frac{|(p_{act} - p_{BPI})|}{p_{act}} \times 100, \qquad (12)$$

where p_{BPI} is the output power of the BPI, p_{act} is the actual leakage reported by HotSpot, and n is the number of samples used in the simulation. Avg. % err. reported was only 10.67% which shows the good leakage modeling of our methodology.

[1]The detailed layout of the LITTLE cores were not given for such layout so we make the evaluation of the whole cluster instead.

TABLE I: New initialization average percentage error versus identity matrix initialization of the three tested floorplans.

Floorplan	Iden. init.	Prop. init.
2×2 mesh	4.4906%	1.7147%
3×3 mesh	2.5568%	1.8863%
big.LITTLE+GPU	5.0425%	3.7715%

2) The proposed initialization verification: To verify the proposed initialization in HotSpot we follow the same methodology used in [7]. In [7] they use both small 2×2 (4 cores) and large 3×3 (9 cores) floorplans. Here, we will use theses two floorplans beside the floorplan shown in Fig. 3 to demonstrate the effectiveness of our approach under different conditions. Table I shows that the proposed approach (Prop. init.) reduces the error in comparison to the case of the identity matrix initialization (Iden. init.) used in [7]. In this experiment, we only used 30 stress patterns, which we found to be adequate for training our BPI models. The results show better accuracy, which is a result of our initialization that better represents the underlying physics of the system and the similarities between different cores as described in Section III.

B. Evaluation using Google Pixel 2 XL

Target Mobile SoC: We choose an Android-based (Oreo version 8.0.0) Google Pixel 2 XL phone as our candidate experimental platform because it has a heterogeneous SoC, which is the Snapdragon-835 octa-core processor that is based on ARM big.LITTLE architecture with two clusters of four big and four LITTLE cores. The maximum operating frequency of the big cores is 2.45 GHz and 1.9 GHz for the LITTLE cluster. In this paper we refer to the four big cores as $big7, big6, big5,$ and $big4$ and for the four LITTLE cores as $LIT3, LIT2, LIT1,$ and $LIT0$. The SoC has a Qualcomm Adreno-540 GPU core with a maximum frequency of 710 MHz. Based on ARM's power management report [15], there are several power saving modes: Standby, Retention, Powerdown, Dormant mode and Hotplug. Only in Powerdown mode the core is completely off and hence any power consumed by the core at that time is considered leakage. In other modes, some parts of the core remain active (e.g. cache RAMs in Dormant mode) and hence the idle power consumed in the other modes is not completely leakage. Since there is no guarantee which one of these modes will be applied when we offline a core, leakage identification cannot be used to identify the leakage in that case. Instead we characterize the power using BPI into active and idle at a fine-grain granularity.

Stress kernels: To provide offline training data for BPI, we generate different stress workload patterns, where we implemented a stress kernel that performs repeated floating-point operations such that the utilization of the cores goes to 100%. The stress patterns are chosen randomly except for the first 8 patterns in which each core is stressed alone keeping the other cores offline, which is useful for the $\tilde{\mathbf{R}}_\mathbf{s}$ and $\tilde{\mathbf{P}}_\mathbf{s}$ initialization, as discussed earlier. For the GPU we use a simple openGL-ES kernel that renders random generated triangles on the screen. While this kernel is running, we

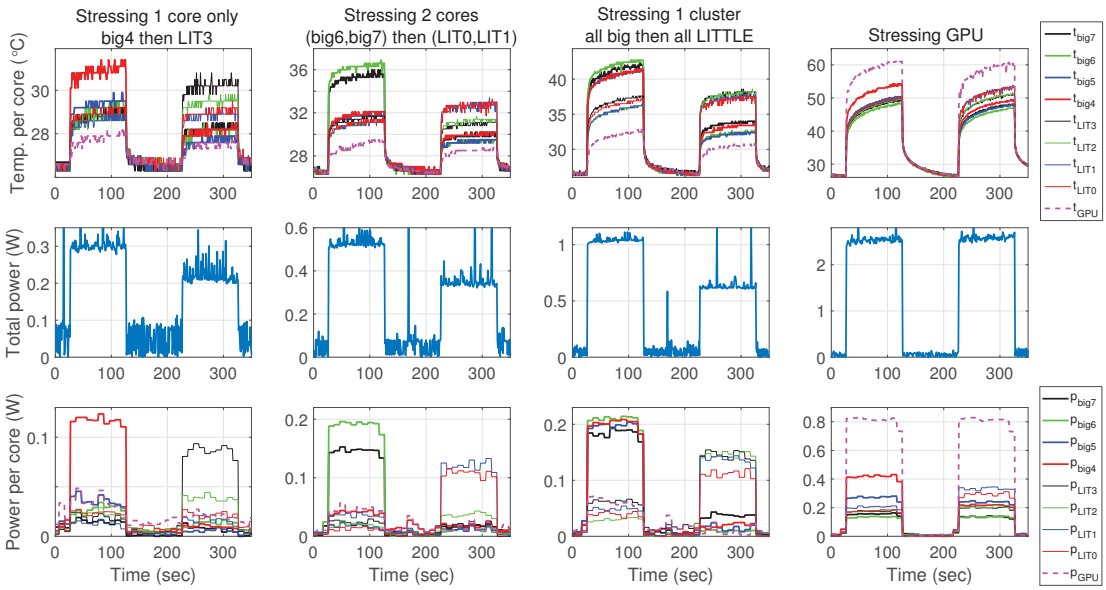

Fig. 5: BPI outputs in different test cases. In the last row the power decomposition in each case is shown.

TABLE II: Summary of fine-grain active and idle power ranges of the big and LITTLE cores. When an entire cluster is idle, the other cluster and GPU are fully active.

	Active Power (mW)	Active temp. (°C)	Idle Power (mW)	Idle temp. (°C)
big4	778.6	76.7	106.2	38.5
big5	793.4	77.7	86.7	35.4
big6	847.6	83.3	132.8	36.4
big7	858.1	83.6	135.4	37.5
LIT0	242.4	42.8	177.0	61.5
LIT1	266.1	43.3	137.4	56.9
LIT2	249.2	43.2	150.0	56.8
LIT3	245.0	43.1	217.3	61.2

set the whole screen color to black, in order to reduce the screen power consumption. Adding to that, the kernel code is modified such that the utilization of the GPU is 100%.

1) BPI evaluation under different benchmarks: Fig. 5 shows the BPI evaluation for big.LITTLE+GPU clusters power decomposition. As shown, the BPI successfully separates the individual power components of each core. In the first trace we stress only core big4 then core LIT3, and it is observed that they have the highest power values in the corresponding power output decomposition of BPI. In the second and third columns subfigures of Fig. 5 we only raised the number of stressed cores to two, then in the third column to four with same conclusions presented in the first case. To reduce the impact of measurement noise, each 10 samples of the power are averaged.

To gain an insight about the power consumption of both big and LITTLE clusters at a fine-grain level, we conduct two experiments in which we stress one cluster at a time plus the GPU while idling the other cluster entirely. The active power numbers of each core reported in Table II represent the average power when its corresponding cluster is stressed. The idle power numbers are average values when the core is idle, and other cluster is stressed. Our results show variations of the active power numbers of the big cluster, e.g. big7 consumes in average 10% more power than big4. The same observation holds with the LITTLE cluster active power, where the variation can be as high as 9%. In average the active power consumption of big cores can be 3.2× higher than the LITTLE cores. Regarding the idle power of both clusters, the average idle power of the LITTLE cluster is 47% higher than the big cluster's idle power. This is due to the high temperatures of the LITTLE cores in idle compared to the temperatures of the big cores, despite that the LITTLE cores have fewer transistors than the big cluster. The mean temperature of the LITTLE cores was 61.5 °C, while it was only 37.5 °C for the big cores. We speculate this higher idle temperature, which lead to higher idle leakage power, for the LITTLE cluster might be due to the floorplan of the tested SoC, where the LITTLE cluster might be located in between or close to both the GPU and the big cluster.

2) Geekbench multi-threaded : To shed light on the power consumption of well-known multi-threaded workloads, we run Geekbench and apply BPI to get the power consumption per core. The data on Table III show that the cryptography workloads consume 15% higher power compared to integer workloads, and it consumes 11% higher power than the floating point workloads. On the other hand the memory workloads consume only 325 mW in average per core. In the meanwhile,

TABLE III: Mean power consumption per core of the big cluster while running Geekbench multi-threaded in mW.

	Cryptography	Integer	Floating point	Memory
big4	930 mW	760 mW	800 mW	410 mW
big5	740 mW	670 mW	690 mW	350 mW
big6	720 mW	680 mW	710 mW	270 mW
big7	750 mW	610 mW	620 mW	270 mW

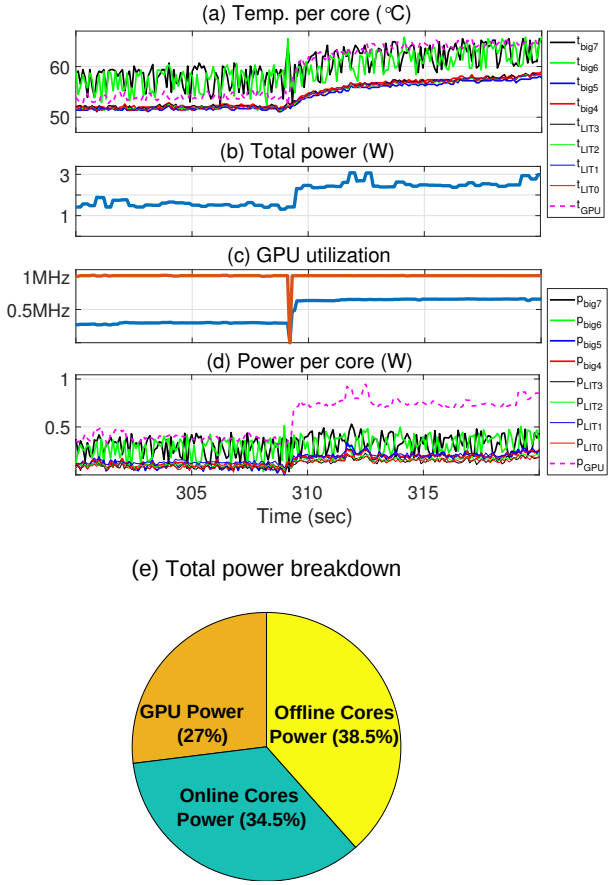

Fig. 6: Active vs. inactive total power components while playing angry-bird.

on the different types of workloads, core big4 seems to have higher power consumption compared to the other cores, for instance for cryptography workloads, in average core big4 consumes 25% more power than the other cores.

3) Power decomposition of angry-bird game: Using BPI we show the power decomposition of angry-birds at fine granularity. We set the playing session for 10 min. Since the GPU needs some online cores to work with, we choose to online cores big6 and big7 and set their frequency to the maximum (2.45 GHz) while offlining all other cores. As shown in Fig. 6.d the GPU always has the highest power component. After 323.5 sec of playing, the game levels become more challenging and then the player faces more interactions with fast movements in the game and this is reflected with more GPU utilization (Fig. 6.c) at the same time. In Fig. 6.e we show the power breakdown between the GPU, the online cores (cores big6 and big7) and the offline cores as a pie chart for more clarification. The breakdown shows how the idle power of offline cores is significant reaching about 38.5% of the total power.

4) Efficiency of big cluster vs. little cluster: To investigate more thoroughly power consumption and its trade-offs in mobile SoCs, we conducted an experiment in which an in-house stress workload is run as a single thread on one of

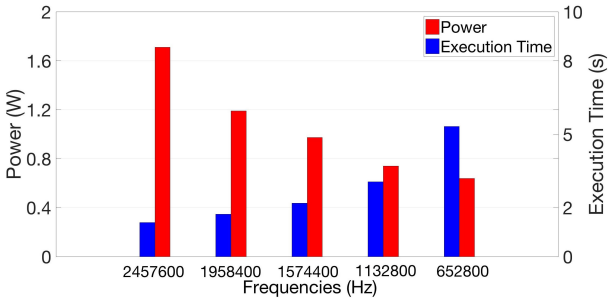

Fig. 7: Power consumption and execution time while running the workload as single threaded on one big core.

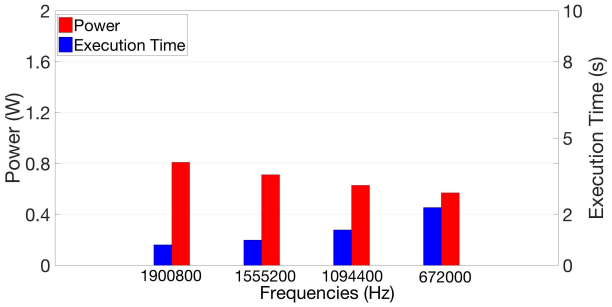

Fig. 8: Power consumption and execution time while running the workload in parallel on all the little cores.

the big cores while keeping track of the power values and performance as measured by execution time. Then these results are compared to the case where the in-house workload is run in parallel on the four cores of the little cluster. Fig. 7 and Fig. 8 show the power and execution time values at different frequencies for single threaded and the multi-threaded case, respectively. The results show interesting numbers, where the performance in the case where the workload is running in parallel on the small cores is improved, and at the same time the power consumption is decreased compared to the single threaded case on a big core. The numbers show that for roughly the same amount of power consumption, the multi-threaded case on the LITTLE cluster results in up to 3× reduction in the execution time. That is, the little cluster can provide in some cases improved runtime and energy efficiency compared to the big cluster.

5) Leakage power under different ambient temperatures: In this experiment we characterize the leakage power of the mobile SoC at different ambient temperatures. First, we control the ambient temperature of the phone by placing it in a thermal chamber. We collect the power traces under 5 different ambient temperatures: 10 °C, 20 °C, 30 °C, 40 °C, and 45 °C. We also set the frequency of both big and LITTLE clusters at the minimum possible frequency (300 MHz) to reduce the voltage to the minimum. As shown in Fig. 9, the increase in temperature is followed by an increase in total power. Since there is no workload running except the light workload of the system kernel, we can consider this increase as a leakage power. The leakage increases from 10 °C to 40 °C is

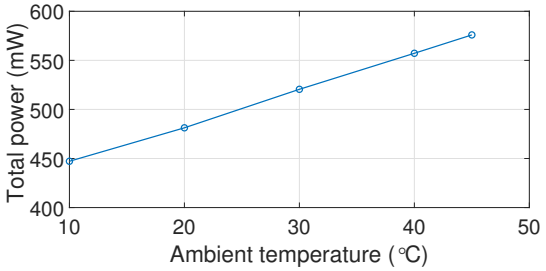

Fig. 9: Total power vs. ambient temperature.

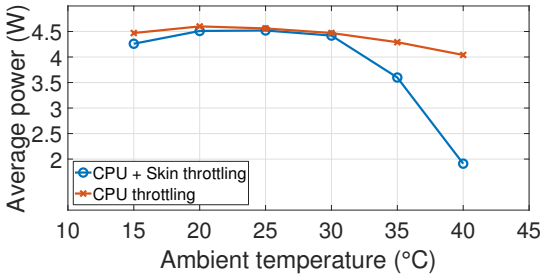

Fig. 10: CPU + Skin vs. CPU throttling on average power.

$\approx$110 mW or $\approx$ 25% increase. This shows how the ambient temperature can significantly affect the power consumption of the phone.

6) Thermal throttling conditions: We also characterize different conditions for thermal throttling. There are two types of throttling: skin throttling (phone skin) and CPU throttling (overheated cores). To study the effect of both kinds accurately we conduct our experiments at different ambient temperatures in the thermal chamber. In our phone, CPU throttling is a built-in feature in Android, whereas skin throttling subroutine is configured from the thermal-engine configuration file when the phone boots and it can be disabled. We use this property to study the effect of throttling on power with only CPU throttling and then with both kinds in action. Firstly, we disable the skin throttling and stress the big cluster, then we measure the average power under 6 different ambient temperatures: 15 °C, 20 °C, 25 °C, 30 °C, 35 °C, and 40 °C. Secondly, we repeat the same experiment but instead we re-enable skin throttling. Fig. 10 reports the average power in the two cases. We observe that for the smallest ambient temperature (15 °C) neither kind of throttling was activated. Then, for 20 °C, only CPU throttling occurs. Starting 30 °C, the ambient conditions and the stress load were enough to cause both skin and CPU throttling. As noticed from Fig. 10, skin throttling is more severe than CPU throttling. We attribute this severe throttling to the thermal-engine configurations where the kernel cap the frequency at only 1.8 GHz, whenever skin thermal sensor exceeds 38 °C.

V. Conclusion

This work gave an insight about the sources of power consumption in mobile SoCs at a fine-grain level, by showing leakage and dynamic power values at the core level, after proposing a novel approach to estimate the leakage power models. We validated the proposed ideas using HotSpot, where the leakage power models of a big.LITTLE+GPU system were estimated at a fine-grain level with an average error of 10%, and the new initialization approach decreased the BPI error from 5.04% to 3.77%. We also implemented and evaluated our method on a real SoC and using real-life applications. Our results show the significance of idle power in mobile platforms, where leakage is becoming a main component of the total power consumption of single cores. Finally, we investigated the active and idle power ranges of the different cores, as well as the trade-offs between power consumption and performance for the big and little clusters.

Acknowledgments: This work was partially supported by NSF grant No. 1438958 and a GRO grant from Samsung Corporation.

References

[1] C. Wang, F. Yan, Y. Guo, and X. Chen. Power estimation for mobile applications with profile-driven battery traces. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 120–125, Sept 2013.

[2] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 105–114, Oct 2010.

[3] W. Liao, J. M. Basile, and L. He. Leakage power modeling and reduction with data retention. In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*., pages 714–719, Nov 2002.

[4] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The design and use of simplepower: a cycle-accurate energy estimation tool. In *Proceedings 37th Design Automation Conference*, pages 340–345, June 2000.

[5] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras. Algorithmic optimization of thermal and power management for heterogeneous mobile platforms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(3):544–557, March 2018.

[6] S. Reda and A. Belouchrani. Blind identification of power sources in processors. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1739–1744, March 2017.

[7] S. Reda, K. Dev, and A. Belouchrani. Blind identification of thermal models and power sources from thermal measurements. *IEEE Sensors Journal*, 18(2):680–691, Jan 2018.

[8] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 359–370, Dec 2006.

[9] F. Beneventi, A. Bartolini, A. Tilli, and L. Benini. An effective gray-box identification procedure for multicore thermal modeling. *IEEE Transactions on Computers*, 63(5):1097–1110, May 2014.

[10] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini. A distributed and self-calibrating model-predictive controller for energy and thermal management of high-performance multicores. In *Design, Automation Test in Europe*, pages 1–6, March 2011.

[11] Y. Xu and W. Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on Imaging Sciences*, 6(3):1758–1789, 2013.

[12] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan. Hotspot: a compact thermal modeling methodology for early-stage vlsi design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, May 2006.

[13] Y. H. Gong, J. J. Yoo, and S. W. Chung. Thermal modeling and validation of a real-world mobile ap. *IEEE Design Test*, 35(1):55–62, Feb 2018.

[14] http://lava.cs.virginia.edu/HotSpot/index.htm. Hotspot version 6.0, 2015.

[15] https://static.docs.arm.com/100960/0100/armv8_a_power_management_100960_0100_en.pdf. Armv8-a power management report, 2017.