Technical report :
Energy consumption modeling and experimental validation on mobile devices

Rapport de recherche :
Modélisation de consommation d'énergie et validation expérimentale sur les appareils mobiles

---

Karel De Vogeleer
Gérard Memmi
Pierre Jouvelot
Fabien Coelho

2013D008

décembre 2013

Département Informatique et Réseaux
Groupe S3 : Systèmes, Logiciels, Services

Rapport de Recherche

Modélisation de Consommation d'Energie et Validation Expérimentale sur les Appareils Mobiles

Karel De Vogeleer[1], Gerard Memmi[1], Pierre Jouvelot[2], and Fabien Coelho[2]

[1] TELECOM ParisTech – INFRES – CNRS LTCI - UMR 5141 – Paris, France
[2] MINES ParisTech – CRI – Fontainebleau, France
{karel.devogeleer,gerard.memmi}@telecom-paristech.fr,
{pierre.jouvelot,fabien.coelho}@mines-paristech.fr

Abstract. Ce papier élabore une preuve théorique et experimentale de l'existence d'une formule de convexité énergie/fréquence. On a opéré sur un smartphone avec un noyau à calcul intensif de plusieurs boucles écrites en C en utilisant un jauge de puissance à haute résolution. Les informations collectées pendant une semaine assument que l'énergie consommée en entrée par unité est forcément corrélée avec la fréquence CPU, et, la courbe fait apparaître un minimum entre 0.2 et 1.6 GHz. On montre que, à part la puissance statique, le comportement assymptotique du temps d'exécution joue un rôle important dans la convexité énergie/fréquence. On fournit un modèle analytique pour ce comportement qui s'adapte bien avec les données. Notre travail doit être intéressant pour les chercheurs qui se concentrent à l'utilisation et la minimisation de l'énergie des systèmes embarquées, et fournit des nouvelles opportunités d'optimisation.

Technical Report

Energy Consumption Modeling and Experimental Validation on Mobile Devices

Karel De Vogeleer[1], Gerard Memmi[1], Pierre Jouvelot[2], and Fabien Coelho[2]

[1] TELECOM ParisTech – INFRES – CNRS LTCI - UMR 5141 – Paris, France
[2] MINES ParisTech – CRI – Fontainebleau, France
{karel.devogeleer,gerard.memmi}@telecom-paristech.fr,
{pierre.jouvelot,fabien.coelho}@mines-paristech.fr

Abstract. This paper elaborates on theoretical and experimental evidence for the existence of an Energy/Frequency Convexity Rule, which relates energy consumption and CPU frequency on mobile devices. We monitored a typical smartphone running a specific computing-intensive kernel of multiple nested loops written in C using a high-resolution power gauge. Data gathered during a week-long acquisition campaign suggest that energy consumed per input element is strongly correlated with CPU frequency, and, more interestingly, the curve exhibits a clear minimum over a 0.2 GHz to 1.6 GHz window. We show that, besides static power, the asymptotic behavior of the execution time plays a large role in the energy/frequency convexity. We provide and motivate an analytical model for this behavior, which fits well with the data. Our work should be of clear interest to researchers focusing on energy usage and minimization for mobile devices, and provide new insights for optimization opportunities.

Keywords: energy consumption and modeling, DVFS, power measurements, execution time modeling, smartphone, bit-reverse algorithm.

1 Introduction

The service uptime of battery-powered devices, e.g., smartphones, is a sensitive issue for nearly any user. This has been shown by academic studies [1] as well as by surveys conducted by retail information channels [2, 3]. Even though battery capacity and performance are hoped to increase steadily over time, improving the energy efficiency of current battery-powered systems is essential because power demands are outpacing the developments in battery capacities. Understanding the energy consumption of the different features of (battery-powered) computer systems is thus a key issue. Providing models for energy consumption can pave the way to energy optimization, by design and at run time.

The power consumption of Central Processing Units (CPUs) and external memory systems is application and user behavior dependent [4]. Moreover, for cache-intensive and CPU-bound applications, or for specific Dynamic Voltage and Frequency Scaling (DVFS) settings, the CPU energy consumption may dominate the external memory consumption [5]. For example, Aaron and Carroll [4] showed that, for an embedded system running `equake`, `vpr`, and `gzip` from the SPEC CPU2000 benchmark suite, the CPU energy consumption exceeds the RAM memory consumption, whereas `crafty` and `mcf` from the same suite showed to be straining more energy from the device RAM memory. Another example is Intel's Data Direct I/O (DDIO) feature, introduced with the E5 Xeon chip, which allows the Ethernet Network Interface Cards (NICs) to load data directly into the CPU's cache [6], minimizing Random Access Memory (RAM) memory

accesses. By avoiding input/output (I/O) operations, the performance is improved but also the energy consumption of the system.

Providing an accurate model of energy consumption for embedded and, more generally, energy-limited devices such as mobile phones is of key import to both users and system designers. To reach that goal, our paper provides both theoretical and first experimental evidence for the existence of an Energy/Frequency Convexity Rule, that relates energy consumption and CPU frequency on mobile devices. This convexity property seems to ensure the existence of an optimal frequency where energy usage is minimal.

This existence claim is based on both theoretical and practical evidence. More specifically, we monitored a Samsung Galaxy SII smartphone running Gold-Rader's Bit Reverse algorithm [7], a small kernel based on multiple nested loops written in C, with a high-resolution power gauge from Monsoon Solutions Inc. Data gathered during a week-long acquisition campaign suggest that energy consumed per input element is strongly correlated with CPU frequency and, more interestingly, that the corresponding curve exhibits a clear minimum over a 0.2 GHz to 1.6 GHz window. We also provide and motivate an analytical model of this behavior, which fits well with the data. Our work should be of clear interest to researchers focusing on energy usage and minimization on mobile devices, and provide new insights for optimization opportunities.

The paper is organized as follows. Section 2 introduces the notions of energy and power, and how these can be decomposed in different components on electronic devices. Section 3 describes the power measurement protocol and methodology driving our experiments, and the C benchmark we used. Section 4 introduces our CPU energy consumption model, and shows that it fits well with the data. Section 5 outlines the Energy/Frequency Rule derived from our experiment and modeling. Related work is surveyed in Section 7. We conclude and discuss future work in Section 8.

2 Power Usage in Computer Systems

The total power P_{total} consumed by a computer system, including a CPU, may be separated into two components:

$$P_{\text{total}} = P_{\text{system}} + P_{\text{CPU}}, \tag{1}$$

where P_{CPU} is consumed by the CPU itself and P_{system} by the rest of system. For Example, P_{system} may include the Global Positioning System (GPS) sensor, consuming power when it is polling for signals, or the AMOLED LCD displays consuming power not only relative to the brightness level as is relative to the kind of color that is displayed [8], and other sources of power consumption, e.g., maintenance of I/O devices (including memory), sensor power supply (gyro-sensors etc). In this work, however, we focus on the power consumption of the CPU (P_{CPU}) and refer to P_{system} as *static* power as we deem these power sources constant over time.

The power consumption P_{CPU} of the CPU can be divided into two parts: P_{dynamic} and P_{leak}, where P_{dynamic} is the power consumed by the CPU during the switching activities of transistors through useful computation. P_{leak} is power originating from leakage effects inherent to silicon-based transistors, and is in essence not useful for the CPU's purposes. P_{dynamic} may be split into the power P_{short} lost when transistors briefly *short-circuit* during gate state changes and P_{charge}, needed to charge the gates' capacitors:

$$P_{\text{dynamic}} = P_{\text{short}} + P_{\text{charge}}. \tag{2}$$

In the literature P_{charge} is usually [9] defined as

$$P_{\text{charge}} = \alpha C f V_{\text{CC}} V_{\text{swing}}, \tag{3}$$

where α is a proportional constant indicating the percentage of the system that is active or switching, C the capacitance of the system, f the frequency at which the system is switching,

V_{CC} the supply voltage, and V_{swing} the voltage swing across C (usually $V_{swing} \approx V_{CC}$, the operating voltage of the CPU).

P_{short} originates during the toggling of a logic gate. During this switching, the transistors inside the gate may conduct simultaneously for a very short time, creating a direct path between V_{CC} and the ground. Even though this peak current happens over a very small time interval, given current high clock frequencies and large amount of logic gates, the short-circuit current may be non-negligible. For example, for a complementary metal-oxide-semiconductor (CMOS) inverter [10] the P_{short} can be approximated by

$$P_{short} = H(V_{CC} - 2V_{TH})^3 \tau f, \tag{4}$$

where H is a technology-dependent scaling factor, τ is the input signal rise time, V_{CC} is the supply voltage, V_{TH} is the threshold voltage, and f is the switching frequency. Formulas of P_{short} for all types of gates are not always available[3] and an aggregated value of P_{short} is tedious to compute for fast clock-rates and complex CPUs. The aggregated effect of P_{short} may, however, be captured by a generalized equation. Given that K, τ, and V_{CC}/V_{TH} are constant, P_{short} is proportional to the switching activity α and the frequency f in Equation 3. A first-order general model may approximate the short-circuit power by deeming it proportional to P_{charge}. Thus the power $P_{dynamic}$ stemming from the switching activities and the short-circuit currents in a CPU is thus

$$\begin{aligned} P_{dynamic} &= P_{charge} + P_{short} \\ &= P_{charge} + (\eta - 1)P_{charge} \\ &= \eta \cdot \alpha C_L f V^2, \end{aligned} \tag{5}$$

where η is a scaling factor representing the effects of short-circuit power.

P_{leak} originates from leakage currents that flow between differently doped parts of a metal-oxide semiconductor field-effect transistor (MOSFET), the basic building block of CPUs. The energy in these currents are lost and do not contribute to the information that is held by the transistor. The leakage current are electrons that move between the transistor's differently doped areas which doesn't directly affect the transistor's amplification characteristics. Some leakage currents are induced during the *on* or *off*-state of the transistor, or both. Six distinct sources of leakage are identified [11]: *reverse-bias pn-junction leakage*, *sub-threshold leakage*, *oxide tunneling current*, *gate current due to hot-carrier injection*, *gate-induced drain leakage*, and *channel punch-through current*. Despite the presence of multiple sources of leakage in MOSFET transistors, the sub-threshold leakage current, gate leakage, and band-to-band tunneling (BTBT) dominate the others for sub-100 nm technologies [12]. Leakage current models, e.g., as incorporated in the Berkeley Short-channel IGFET Model (BSIM) [11] micro models, are accurate, nevertheless complex since they depend on multiple variables. Moreover, P_{leak} fluctuates constantly as it also depends on the temperature of the system. Consequently P_{leak} cannot be considered a static part of the system's power consumption. Given the different sources of power consumption in a MOSFET-based CPU, the total power can be rewritten as

$$P_{total} = P_{system} + P_{leak} + P_{dynamic}. \tag{6}$$

The leakage power and short-circuit power loss can be minimized, e.g., by superior Very-Large-Scale Integration (VLSI) technology, by voltage gating of function units, memory, and cache [13, 14]. For example You *et al.* [15] extended an Instruction Set Architecture (ISA) with instructions to support the control of power-gating at the component level. Via simulations the authors claim to reduce the power consumption of their benchmarks on the average by 23.05%.

Figure 1 provides an overview of the power breakdown that is assumed in this work.

[3] The physical properties and dimensions of transistors and gates are not always publicly available. Therefore we must often resort to macro-level modeling.

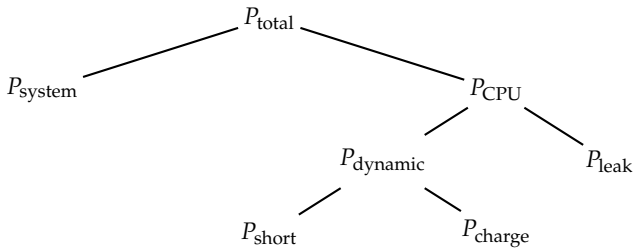


Fig. 1. Power breakdown of a computer system as assumed in this work.

The relationship between the *power* $P(t)$ (Watts or Joules/s) and the *energy* E (Joules) consumed by an electrical system over a time period Δt is given by

$$E = \int_0^{\Delta t} P(t)\, dt = \int_0^{\Delta t} I(t) \cdot V(t)\, dt, \tag{7}$$

where $I(t)$ is the current supplied to the system, and $V(t)$ the voltage drop over the system. Often $V(t)$ is constant over time, hence $dP(t)/dt$ only depends on $I(t)$. If both *current* and *voltage* are constant over time, the energy integral becomes the product of *voltage*, *current* and *time*, or alternatively *power* and *time*:

$$E = \int_0^{\Delta t} P(t)\, dt = V \cdot I \cdot t. \tag{8}$$

3 Power Measurement Protocol on Mobile Devices

A Samsung Galaxy S2 is used in our testbed sporting the Samsung Exynos 4 Systems-on-Chip (SoC) 45 nm dual-core. The Galaxy S2 has a 32 KB L1 data and instruction cache, and a 1 MB L2 cache. The mobile device runs Android 4.0.3 on the Siyah v3.2.6.4 kernel adopting Linux 3.0.31. The frequency scaling governor in Linux was set to operate in *userspace* mode to prevent frequency and voltage scaling on-the-fly. The second CPU core was disabled during measurements. The smartphone is booted in *clockwork recovery* mode to minimize noisy side-effects of the Operating System (OS) and other frameworks.

During the experiments, the phone's battery was replaced by a power supply (Monsoon Power Monitor) that measures the power consumption at 5 kHz with an accuracy of 1 mW. The power of the system and the temperature of the CPU were simultaneously logged. The CPU temperature traces were collected via the Exynos 4 build-in temperature sensor. The noise of the temperature sensor shows an interquartile range (IQR) of 0.45°C. The temperature samples are accessible through the `/proc` file system. Also, the temperature is written periodically to the kernel debug output with a default frequency of 1 sample per 30 s. We opted to patch the kernel to print the temperature samples to the kernel output at a rate of 2 Hz and read out the samples from there to obtain a milli-second timestamp accuracy. This was achieved by amending the `sec_therm_pdata` struct in the `arch/arm/mach-exynos/mach-u1.c` file. Figure 2 shows a picture of the measurement setup. We stripped off the phone of as many components as possible, e.g., the camera, GPS, and liquid crystal display (LCD) display, to reduce noise in the power measurements. A computer fan, originally mounted on a Pentium 4 heat sink, was used to control the temperature of the testbed by positioning the CPU under the spinning blades of the cooling device. Note that the fan was able to cool down the CPU, working at 100% utilization, for frequencies under 1.2 GHz. For larger frequencies the CPU heat generation was greater than the cooling capabilities of the computer fan.

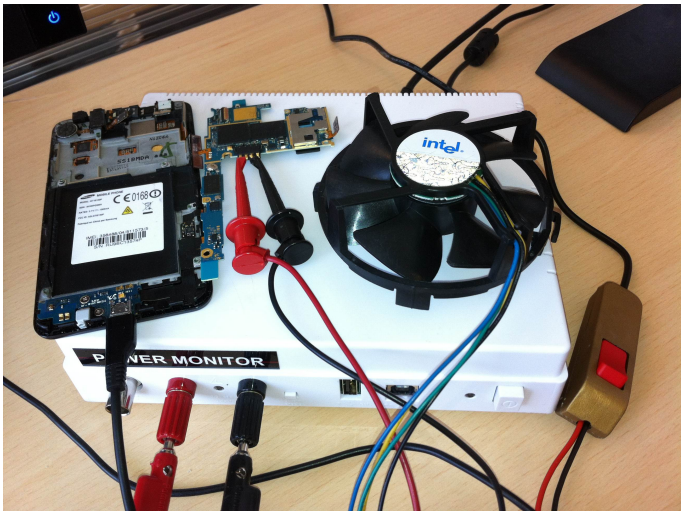


Fig. 2. Measurement setup showing the power gauge (below), the disassembled Samsung Galaxy S2 (left), the fan (right), and its switch, used to control the temperature of the testbed.

The *bit-reverse* algorithm is used as benchmark kernel. This is an important operation since it is part of the ubiquitous Fast Fourier Transformation (FFT) algorithm, and rearranges deterministically elements in an array. The bit-reversal kernel is CPU intensive, induces cache effects, and is economically pertinent. The Gold-Rader implementation of the bit-reverse algorithm, often considered the reference implementation [7], is given below:

```
void bitreverse_gold_rader (int N, complex *data) {
    int n = N;
    int nm1 = n-1;
    int i = 0, j = 0;

    for (; i < nm1; i++) {
        int k = n >> 1;
        if (i < j) {
            complex temp = data[i];
            data[i] = data[j];
            data[j] = temp;
        }
        while (k <= j) {
            j -= k;
            k >>= 1;
        }
        j += k;
    }
}
```

The input of the bit-reversal algorithm is an array with a size of 2^N; the elements are pairs of 32 bit integers, representing complex numbers. Note that array sizes up to 2^9 fit in the L1 cache, while sizes over 2^{18} are too big to fit in the L2 cache. During the measurements, N is set between 6 and 20 in steps of 2, while varying the CPU frequency from 0.2 GHz to 1.6 GHz in steps of 0.1.

To minimize overhead, 128 copies of the kernel are run sequentially. For time measurement purposes, this benchmark is repeated 32 times for at least 3 seconds each time (this may require multiple runs of the 128 copies). For the power and temperature measurements, the benchmark is repeated in an infinite loop until 32 samples can be gathered. The benchmark is compiled with GCC 4.6, included in Google's NDK, generating ARMv5 thumb code.

The power samples were measured at a temperature of 37°C. The CPU's temperature was therefore oscillated between 36°C and 38°C such that the power could be logged every time the temperature passed 37°C. To achieve the temperature oscillation our fan was turned *on* and *off* for frequencies below 1.2 GHz. For frequencies above, the fan was unable to cool down the CPU. Consequently, the temperature was controlled by turning the testbed *on* and *off*, while the fan was spinning.

4 Modeling Energy Consumption

Energy is the product of time by power. We look at each of these factors in turn here.

4.1 Execution Time

Since applications run over an OS on top of hardware, we need to take account for those when estimating computing time. Indeed, an OS needs a specific amount of time, or clock cycles, to perform (periodical) tasks, e.g., interrupt handling, process scheduling and processing kernel events. The hardware halts the execution of software sometimes momentarily, e.g., during pipe-line stalls or memory stalls. When the processor is not spending time in kernel mode and the software is not halted, the processor is available for user-space programs, e.g., our benchmark. From a heuristic point of view, it can be assumed that the OS kernel and hardware need a fixed number of clock cycles cc_k per time unit to complete their tasks. Thus, we propose to model the amount of clock cycles to complete a benchmark sequence of instructions cc_b as

$$cc_\mathrm{b} = t(f^\beta - cc_\mathrm{k}), \tag{9}$$

where cc_k are the number of clock cycles spent in the OS and on hardware halts, t the total time needed to complete the program, f the system's clock frequency and β an architecture-dependent scaling constant, to be fitted later on with the data. The definition of cc_b is rewritten to isolate the execution time:

$$t = \frac{cc_\mathrm{b}}{f^\beta - cc_\mathrm{k}}. \tag{10}$$

Note that t tends to zero for $f \to \infty$ and there is a vertical asymptote at $\sqrt[\beta]{cc_\mathrm{k}}$. This formulation of time can also be found in the work of Snowdon [16] (Equation 1) though presented differently and used in a different context. We refer to cc_k and $\frac{cc_\mathrm{b}}{f}$ whereas Snowdons refers to $C_\mathrm{tick} f_\mathrm{tick}$ and T_work, respectively.

Figure 3 shows the execution time of the benchmark for different clock frequencies. The dotted lines are the data fitted on Equation 10. The data was fitted using the `nls()` function in R [17], with the `port` algorithm enabled. Table 3 shows the fitting errors of Equation 10 on the execution time measurement data, averaged over all tested input sizes. The fitting shows a vertical asymptote around 115 MHz. This may indicate the minimum amount of clock cycles required by the OS of the device to operate. The measurement data for input sizes 2^6 up to 2^{16} are well described by Equation 10. However, sizes 2^{18} and 2^{20}, too large to fit within cache L2, seem to operate under different laws. Therefore, from now on, the attention is focused on data that fit in the cache of the CPU.

4.2 Power Consumption

If dynamic power modeling is rather easy (see §2), the case for leakage is more involved, and warrant a longer presentation. In particular, leakage power is heavily temperature-dependent [11]. For example, our CPU at 1.3 GHz shows an inflated power consumption of around 5% between a CPU temperature of 36°C and 46°C. You *et al.* [15] shows similar results for a 0.1μm processor; a temperature increase from 30°C to 40°C leads to a 3% power increase. On the other hand, the power P_charge required for a given computation does not change with regards to the CPU temperature. Given that the system's supply voltage and P_system is constant, it is the increasing leakage current I_leak that inflates the power consumption. The leakage current is dependent on the temperature. The three most notable effects that constitute the leakage current in MOSFET transistors are; the sub-threshold current, the gate leakage current, and the BTBT leakage. The BSIM shows that the leakage current micro models depend on a multitude of variables. The models of these currents contain variables that, inter alia, are technology-specific, e.g., transistor dimensions, gate oxide capacitance. The transistor's terminal voltages play also an important role. The temperature itself appears

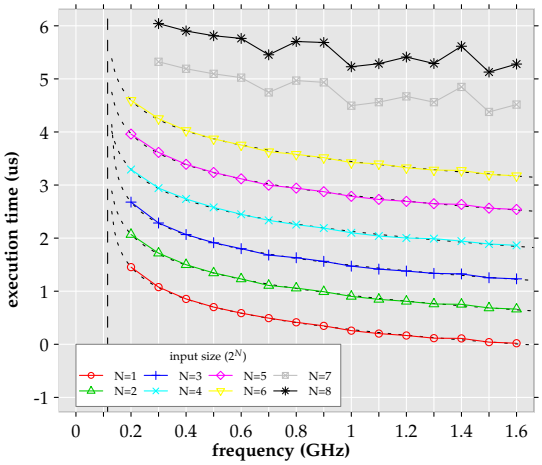


Fig. 3. Experimentally-measured *execution time* of the benchmark given the system's clock frequency. The dotted lines represent the fitted curves on Equation 10. The average vertical asymptote $\sqrt[\beta]{cc_{\mathrm{k}}}$ is shown around 115 MHz.

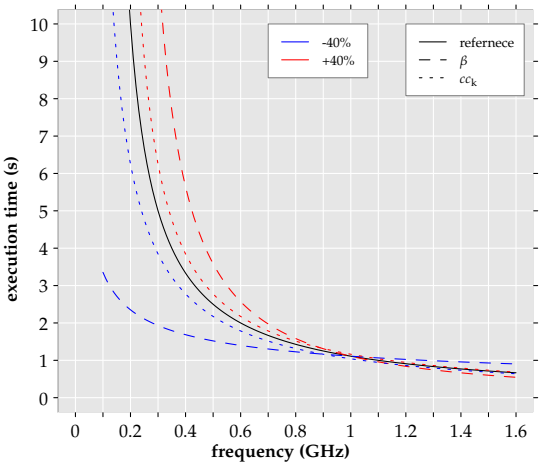


Fig. 4. Impact of the parameters β and cc_{k} on the time definition $t = cc_{\mathrm{b}}/(f^{\beta} - cc_{\mathrm{k}})$. (Default values $\beta = 1$, $cc_{\mathrm{k}} = 1$, and $cc_{\mathrm{b}} = 1$)

several times in the sub-threshold and BTBT leakage models; the gate leakage however is not temperature dependent. Mukhopadhyay *et al.* [18] showed via simulation that for 25 nm technology the sub-threshold leakage current is dominant over the BTBT leakage current, but the latter cannot be neglected.

Under forced cooling, the CPU temperature T at time t can be approximated with Newton's Law of cooling [19] where $T(t)$ approaches T_e for $t \to \infty$. Here T_e is the equilibrium temperature at which the heat generation of the CPU equals the heat convection of the forced cooling:

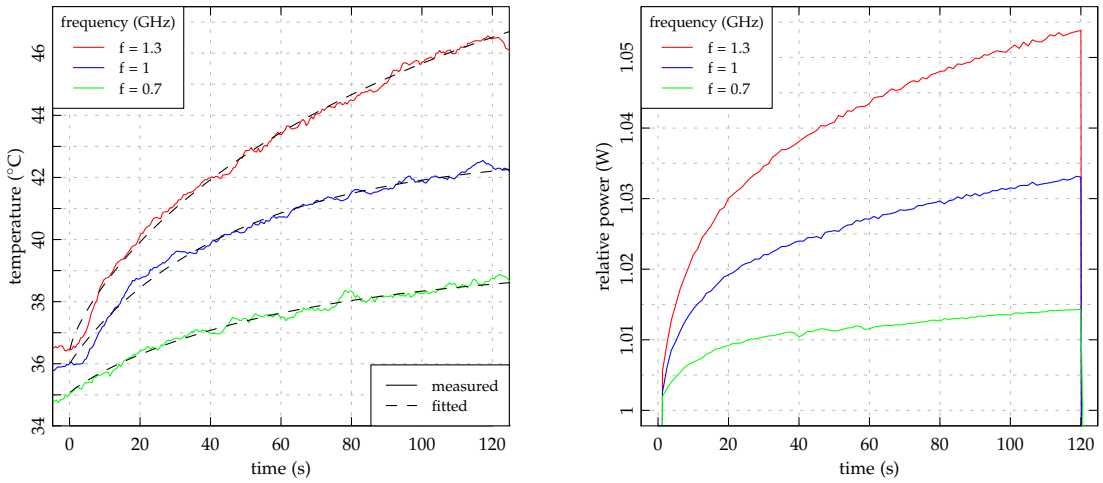
$$\begin{aligned} \frac{dT}{dt} &= \bar{\alpha}(T_e - T(t)) \\ T(t) &= T_e - (T_e - T_0)e^{-\bar{\alpha}t}, \end{aligned} \tag{11}$$

where T_0 is the temperature at $t = 0$, T_e is the equilibrium temperature. The dotted lines in Figure 5 represent the temperature fitted on Equation 11. Unfortunately, in these traces the current temperature measurements are too noisy to model the temperature by fitting its derivative, and providing statistically-significant results.

Under normal conditions, the temperature of the CPU's silicon varies continuously depending on the load of the CPU and the system's ambient temperature. Therefore, to have a fair comparison of energy consumption between different code pieces one needs to compare the measurements at a reference temperature. In our measurements the CPU temperature difference of 10°C increases the power by about 7% in the worst case scenario. Ideally, to model the temperature dependence of the power, a formulation for I_{leak} is needed where the temperature T can be isolated from the other variables: $I_{\mathrm{leak}}(T, \cdot) = f(T)f(\cdot)$. In such a formulation $f(T)$ is a scaling factor of $f(\cdot)$, and hence of I_{leak}, that is independent of the transistor's technicalities, e.g., state or physical dimensions. Finding a temperature scaling factor for the leakage current is however not a straightforward task. Nevertheless, approximative scaling factors have been analytically obtained or experimentally defined via simulations (mainly SPICE):

- Su *et al.* [20]: $\dfrac{\Delta I_{\mathrm{leak}}}{I_{\mathrm{leak}}} = 1 + a_1\Delta T + a_2\Delta T^2$,
- Liao *et al.* [21]: $I_{\mathrm{leak}} = I_0 T^2 e^{a_1/T}$,
- Liao *et al.* [22]: $I_{\mathrm{leak}} = I_0 e^{-a_1/(T-a_2)}$,
- Ferre *et al.* [23] and Sinha *et al.* [24]: $I_{\mathrm{leak}} = I_0 e^{a_1/T}$,

where I_0 is the leakage current at a given reference temperature, and $a_{\cdot}$ is a constant. After analysis on our data, we discovered that none of the cited approximations would fit well. This is because the rationale



(a) time trace (b) power trace

Fig. 5. Power (b) and temperature (a) traces of a constant CPU workload, run for 120 seconds, at different CPU clock frequencies. The power trace is relative to the power at time 0. For the temperature trace, the dotted lines show the fitting on Newton's law of cooling.

Table 1. Frequency and voltage relationship for the Dynamic Voltage and Frequency Scaling (DVFS) process in the default Siyah kernel.

f (MHz)	25	50	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600
V (mV)	850	875	900	925	950	950	950	975	1000	1025	1075	1125	1175	1225	1250	1275	1325	1350

on which these approximations are based assume conditions, which are not entirely realistic, to simplify the leakage current micro models. Most previous research works focus solely on the sub-threshold leakage effect, neglecting other leakage effects. This may be appropriate for technologies larger than the 45 nm technology we use. Moreover, these models require the knowledge of multiple parameters and I_0 which is not always available at hand or on the fly. Skadron *et al.* [25] studied the temperature dependence of I_{leak} as well. Skadron *et al.* deducted a relationship between the leakage power P_{leak} and dynamic power P_{dynamic} based on International Technology Roadmap for Semiconductors (ITRS) measurement traces (variables indexed with 0 are reference values):

$$R_T = \frac{P_{\text{leak}}}{P_{\text{dynamic}}} = \frac{R_0}{V_0 T_0^2} e^{\frac{B}{T_0}} V T^2 e^{\frac{-B}{T}}. \tag{12}$$

If the temperature T is stable across different operating voltages, then the value of R_T is a function of V multiplied by a constant, lets say γ, which includes the temperature-dependent variables and other constants. Total power P_{total} is thus:

$$\begin{aligned} P_{\text{total}} &= P_{\text{system}} + P_{\text{leak}} + P_{\text{dynamic}} \\ &= P_{\text{system}} + \gamma V P_{\text{dynamic}} + P_{\text{dynamic}} \\ &= P_{\text{system}} + (1 + \gamma V) \cdot \eta \alpha C f V^2. \end{aligned} \tag{13}$$

This formulation of P_{total} incorporates three parameters: P_{system}, γ, and $\eta\alpha C$. Experimental values of these variables can be obtained via fitting power traces on Equation 13. V and f are linked via the DVFS process inherent to the Linux kernel and the hardware technicalities. The values for our CPU are found inside the kernel source code; they are shown in Table 1 and plotted in Figure 6.

Equation 13 is fit on our benchmark power measurements. The power fitting errors are shown in Table 3 for a 37°C CPU temperature. The fitting errors are on the average not larger than 3 % except for the measurement point at 1.5 GHz. This measurement point was obtained at different independent occasions but appears, for obscure reasons, to disobey persistently the model in Equation 13.

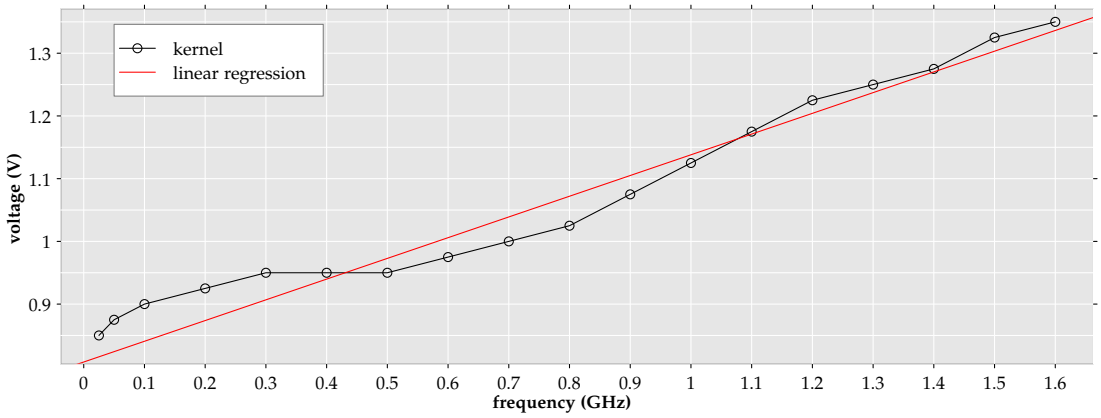


Fig. 6. Visualization of the Frequency and voltage relationship for the Dynamic Voltage and Frequency Scaling (DVFS) process in the default Siyah kernel.

Given a linear relationship between f and V Equation 13 can be expanded, resulting a 4th order polynomial

$$P_{\text{total}} = a_4 f^4 + a_3 f^3 + a_2 f^2 + a_1 f^1. \tag{14}$$

It is noted that this formulation does not have a constant term a_0. Equation 13 is sometimes approximated by a third-order polynomial. This is referred to as the *cube root rule* [14]: $P_{\text{total}} \approx K_v V^3 \approx K_f f^3$, where K_v and K_f are constants. Equation 13 however suggests a relationship between P_{total} and V or f of the fourth order. Comparing fitting errors of a fourth order polynomial and lower order polynomials indeed support the claim that for our testbed the 4th-order is most appropriate.

4.3 Energy Consumption

Typical compute-intensive programs incur approximately a constant load on the CPU and system, barring user interactions. Moreover, if the time to complete one program is also much smaller than the sampling rate of the power gauge, then $P(t)$ in Equation 7 is constant. Hence, it suffices to sample the power P_{bench} of a benchmark at a given CPU temperature and multiply this value by the execution time of the benchmark t_{bench} to get an energy estimate. As a result the definition of time in Equation 10, and power in Equation 13, can be used to model the energy of one benchmark kernel. The energy consumed by the CPU E_{CPU} is given by

$$\begin{aligned} E_{\text{CPU}} &= E_{\text{leak}} + E_{\text{dynamic}} \\ &= P_{\text{bench}} \cdot t_{\text{bench}} \\ &= \left((1 + \gamma V) \cdot \eta \alpha C f V^2 \right) \cdot \frac{cc_{\text{b}}}{f^{\beta} - cc_{\text{k}}}. \end{aligned} \tag{15}$$

Constants γ, $\eta\alpha C$, cc_{b}, and cc_{k} in this formulation were evaluated before via fitting.

5 The Energy/Frequency Convexity Rule

Using the testbed and models described above, Figure 7 shows the measured and modeled energy E_{CPU} for our benchmark kernel over the different frequencies; data have been normalized over the benchmark input size. Table 3 shows the average absolute energy error between our fitted model and the measured data. The average fitting error stays below 6 % except for measurement points 1.5 GHz and 200 MHz. The large fitting error in the 200 MHz case stems from the large execution time that amplifies the power measurement fitting error (see Table 3). It can also be seen that, for larger benchmark input sizes, on the average more energy is required. This is the result of higher level cache utilization.

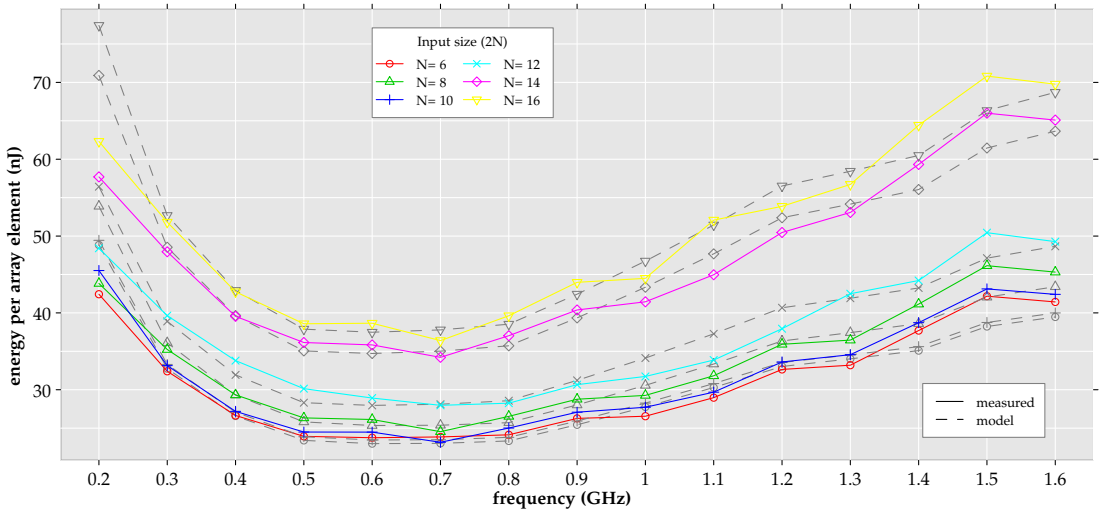


Fig. 7. Energy required by the CPU at 37°C to complete our benchmark kernel given an input size. The dashed lines denote the theoretical curve as per Equation 15.

Figure 7 exhibits a clear convex curve, with a minimum at Frequency f_{opt}, suggesting the existence of an Energy/Frequency Convexity Rule for compute-intensive programs. Why is the energy consumption curve convex? The energy consumption of the benchmark kernel scales approximately linearly with the number of instructions (see further). The time Δt it takes to execute an instruction sequence increases more than linearly with decreasing operating frequency (asymptotic behavior, see Figure 6). P_{leak} is independent of the type of computation, and E_{leak} builds up linearly with time: $E_{\text{leak}} = P_{\text{leak}}\Delta t$. P_{leak} becomes increasingly important in the part where the CPU frequency f is smaller than f_{opt}. For the part where $f > f_{\text{opt}}$, the inflated E_{CPU} can be attributed to the increasing supply voltage V_{CC} affecting P_{dynamic} quadratically. Furthermore, had P_{system} been incorporated in the picture, then f_{opt} would have moved to a higher frequency because the additional consumed energy of the system could have been minimized by a faster completion of the computations on the CPU.

Our proposal for the existence of an Energy/Frequency Convexity Rule can be further supported using our previous models. Indeed, we can model the relationship in Table 1 between the *frequency* (GHz) and *voltage* (V) in the Linux kernel with a linear approximation:‘

$$\begin{aligned} V &= m_1 f + m_2, \\ &= 0.3304 \cdot f \cdot 10^{-9} + 0.8077. \end{aligned} \tag{16}$$

Now the derivative of E_{CPU} defined in Equation 15 over f or V can be computed. The energy curve shows a global minimum $E_{\text{CPU,min}}$ for f_{opt} when its derivative is equal to zero ($\partial E_{\text{CPU}}/\partial f = 0$) and its second derivative is positive ($\partial^2 E_{\text{CPU}}/\partial f^2 > 0$). The formal definition of strict convexity is given by

$$E_{\text{CPU}}(tf_1 + (1-t)f_2) < tE_{\text{CPU}}(f_1) + (1-t)E_{\text{CPU}}(f_2), \tag{17}$$

where $t \in \{0,1\}$. We verified this inequality via a Monte Carlo simulation over the interval $0.2\,\text{Ghz} < f < 1.6\,\text{Ghz}$. Within these boundaries it is shown that Equation 15 is strict convex whenever $0.2 > \sqrt[\beta]{cc_{\text{k}}}$.

Given that E_{CPU} only shows one minimum, f_{opt} is the global minimum if the following equality holds:

$$\frac{(1+\gamma V)Vf^{\beta}\beta}{f^{\beta} - cc_{\text{k}}} = fm_1(3\gamma V + 2) + (1+\gamma V)V. \tag{18}$$

Four parameters appear in this formulation that affect the optimal frequency f_{opt}: β and cc_{k}, which are related to the execution time of the benchmark, m_1 the slope between V and f, and γ related to the leakage current ratio. Simulations show that, if β or cc_{k} decreases, $E_{\text{CPU,min}}$ will shift to a higher frequency, and, if

Table 2. Relative influence of the parameters β, cc_k, γ, and m_1 on f_{opt}. The reference E_{CPU} is calculated given $\beta = 1.27$, $\gamma = 3.14$, $cc_b = 100$, $\eta\alpha C = 0.34$ and $cc_k = 0.07$. The parameters are then scaled and the relative change in f_{opt} is then reported.

	-75%	-50%	-25%	-10%	-5%	-1%	+1%	+5%	+10%	+25%	+50%	+70%	+100%
β	0.37	0.37	0.48	0.78	0.89	0.98	1.02	1.11	1.24	1.63	2.46	2.96	2.96
cc_k	0.67	0.80	0.91	0.96	0.98	1.00	1.00	1.02	1.04	1.09	1.17	1.22	1.31
m_1	2.96	1.69	1.22	1.07	1.04	1.00	1.00	0.96	0.94	0.87	0.78	0.72	0.67
γ	1.09	1.04	1.02	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98	0.98	0.98

m_1 or γ decreases, $E_{CPU,min}$ will decrease as well. γ is temperature dependent; if the temperature increases, γ will increase accordingly. Hence, f_{opt} increases with temperature as well. Table 2 shows the relative influence of the parameters β, cc_k, γ, and m_1 on f_{opt}. For the presented measurements Equation 15 shows a minimum on the average around 700 MHz. This holds for all input sizes of the benchmark between 2^6 and 2^{16}. As a result, this implies that there exists an operating frequency, which is neither the maximum nor the minimum operating frequency, at which the CPU would execute a code sequence on the top of the OS in the most energy efficient way.

6 Energy and Execution Time

Figure 6 shows the relationship between energy consumption and the execution time of the benchmark, based on Equation 15. It can be observed that for this benchmark the energy consumption and the execution time exhibit a linear relationship. This implies that performance optimization is the same as optimizing for energy in this case. Such findings have been pointed out before in the literature. Pallister *et al.* [26] showed via different compiler settings that the energy needed to execute a sequence of code is approximately linearly proportional to the time needed to execute that sequence. Bellosa [27] showed similar measurements for integer and floating point operations. This is in line with the observation that the energy needed to execute a single instructions doesn't differ largely [24, 28]. The total CPU energy consumption however also depends on the length and the state of the pipeline, inter-instruction Hamming distance, number of operands, or the inter-instruction effects [28]. A random sequence of instructions will nonetheless yield a total energy consumption that is approximately proportional to the number of instructions in the sequence.

We point out that increasing the CPU frequency from 0.2 GHz to 0.4 GHz speeds up the program by a factor of 4, whereas a frequency increase from 0.8 GHz to 1.6 GHz improves execution time by a factor of 2.5. Thus halving the execution time requires an increasing frequency need. Moreover, the energy required to sustain this frequency increase becomes relatively increasingly larger than the time gain. For example, for an input size of 4 KiB, to speed-up by three-fold between 0.7 GHz and 1.6 GHz 1.46 times more energy is required. Similarly, to speed up the benchmark three-fold from 1.6 GHz the processor needs to be theoretically clocked at 3.6 GHz, that would require 1.9 times more energy per array element. These conclusion can also be deducted from Equation 15.

7 Related Work

The convex property of the energy consumption curve has been hinted at before in the literature. A series of papers, approaching the problem from an architectural point of view, have shown a convex energy consumption curve with respect to DVFS [5, 29, 30]. The authors put forward some motivation, but do not provide an analytical framework. Other studies, e.g., Hager *et al.* [31] and Freeh *et al.* [32], discuss what the consequences are of said behaviour and how to exploit it, from a high-level point of view. Other research have also shown energy measurements under DVFS processes but no convexity is shown by the measurements, e.g., Sinha and Chandrakasan [24], and Šimunić *et al.* [33] who are not running their benchmark on top of an OS.

In the VLSI design domain voltage scaling has also been discussed but usually for a fixed frequency [34, 35]. The aim of the voltage scaling is to find a minimum energy operation point where the digital circuit yields the correct output. The major trade-off is between increasing circuit latency and leakage power, and

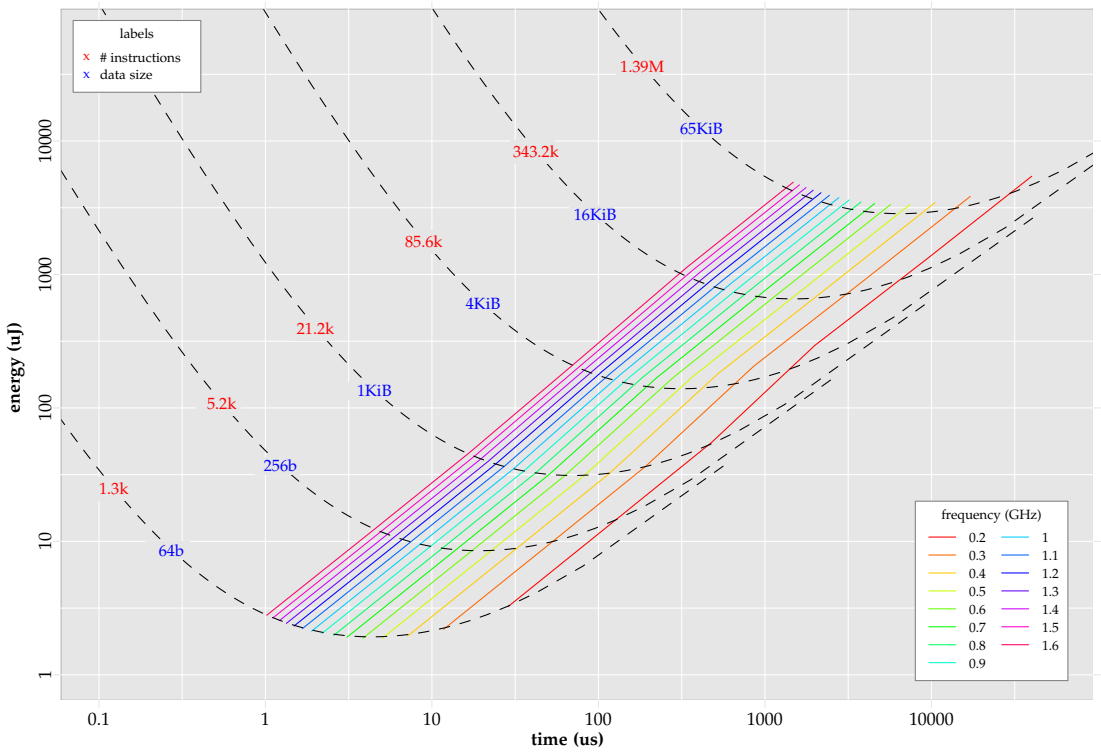


Fig. 8. Energy consumption and execution time relationship for different frequencies. A dashed lines is the benchmark performance with a given input size (data size).

decreasing dynamic power. This trade-off yields also a convex energy consumption curve, but for a fixed frequency.

8 Conclusion

We provide an analytical model to describe the energy consumption of a code sequence running on top of the OS of a mobile device. The energy model is parametrized over five parameters abstracting the specifics of the Dynamic Voltage and Frequency Scaling (DVFS) process, the execution time related parameters, and the power specifications of the CPU. Measurement traces from a mobile device were used to validate the appropriately fitted model. It is shown that the model is on the average more than 6 % accurate. The importance of power samples obtained at a reference temperature is also pointed out.

It is also shown that the analytical energy model is convex (representing what we call the Energy/Frequency Convexity Rule) and yields a minimum energy consumption of a code sequence for a given CPU operation frequency. This minimum is a function of the temperature, execution time related parameters, and technical parameters related to the hardware.

Future work includes checking the validity of our model and its parameters over a wide range of compute-intensive benchmarks. Also, extending the presented model to better handle memory access operations, in particular the impact of caches, is deserved. Finally a generalization of the model to encompass the impact of other programs running in parallel with benchmarks or system power effects would be useful.

Acknowledgements

We would like to thank Amal Ellouze for her interesting grammatical discussions while drafting this document.

References

1. S. Ickin, K. Wac, M. Fiedler, L. Janowski, J.-H. Hong, and A. Dey, "Factors influencing quality of experience of commonly used mobile applications," *Communications Magazine, IEEE*, vol. 50, no. 4, pp. 48 –56, april 2012.
2. CCN.com. (2005, September) Battery life concerns mobile users. [Online]. Available: www.cnn.com/2005/TECH/ptech/09/22/phone
3. J. Paczkowski. (2009, Augustus) Iphone owners would like to replace battery. [Online]. Available: http://digitaldaily.allthingsd.com/20090821/iphone-ownerswould-like-to-replace-battery-att
4. A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proceedings of the USENIX conference on USENIX*, Berkeley, CA, USA, 2010.
5. D. C. Snowdon, S. Ruocco, and G. Heiser, "Power management and dynamic voltage scaling: Myths and facts," in *2005 WS Power Aware Real-time Comput.*, New Jersey, USA, Sep 2005.
6. Intel Corp., "Intel Data Direct I/O Technology Overview," 2012. [Online]. Available: http://www.intel.ie/content/dam/www/public/us/en/documents/white-papers/data-direct-i-o-technology-overview-paper.pdf
7. B. Gold and C. M. Rader, *Digital processing of signals*. McGraw-Hill NY, 1969.
8. X. Chen, Y. Chen, Z. Ma, and F. C. A. Fernandes, "How is energy consumed in smartphone display applications?" in *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '13. New York, NY, USA: ACM, 2013, pp. 3:1–3:6.
9. N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI design: a systems perspective*. Boston, USA: Addison-Wesley Longman Publishing Co., Inc., 1985.
10. H. Veendrick, "Short-circuit dissipation of static cmos circuitry and its impact on the design of buffer circuits," *Solid-State Circuits, IEEE Journal of*, vol. 19, no. 4, pp. 468–473, aug 1984.
11. W. Liu, X. Jin, K. Kao, and C. Hu, "BSIM 4.1.0 MOSFET model-user's manual," EECS Dept., Univ. of California, Berkeley, Tech. Rep. UCB/ERL M00/48, 2000.
12. A. Agarwal, S. Mukhopadhyay, C. Kim, A. Raychowdhury, and K. Roy, "Leakage power analysis and reduction: models, estimation and tools," *Computers and Digital Techniques, IEEE Proceedings -*, vol. 152, no. 3, pp. 353–368, may 2005.
13. K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, no. 2, pp. 305 – 327, feb 2003.
14. Y. N. Srikant and P. Shankar, Eds., *The Compiler Design Handbook: Optimizations and Machine Code Generation*. CRC Press, 2008.
15. Y.-P. You, C. Lee, and J. K. Lee, "Compiler analysis and supports for leakage power reduction on microprocessors," in *Proceedings of the 15th international conference on Languages and Compilers for Parallel Computing*, ser. LCPC'02. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 45–60.
16. D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser, "Koala: a platform for OS-level power management," in *Proceedings of the 4th ACM European conference on Computer systems*, ser. EuroSys '09. New York, NY, USA: ACM, 2009, pp. 289–302.
17. R. Gentleman and R. Ihaka. (2013, June) The R Project for Statistical Computing. [Online]. Available: http://www.r-project.org/
18. S. Mukhopadhyay, A. Raychowdhury, and K. Roy, "Accurate estimation of total leakage current in scaled cmos logic circuits based on compact current modeling," in *Design Automation Conference, 2003. Proceedings*, june 2003, pp. 169 – 174.
19. M. Gockenbach and K. Schmidtke, "Newton's law of heating and the heat equation," *Involve Mathematical Journal*, vol. 2, no. 4, pp. 439–450, Oct. 2009.
20. H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif, "Full chip leakage estimation considering power supply and temperature variations," in *Proceedings of the 2003 international symposium on Low power electronics and design*, ser. ISLPED '03. New York, NY, USA: ACM, 2003, pp. 78–83.

21. W. Liao, L. He, and K. M. Lepak, "Temperature and supply voltage aware performance and power modeling at microarchitecture level," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 24, no. 7, pp. 1042–1053, Nov. 2006.
22. W. Liao, J. M. Basile, and L. He, "Leakage power modeling and reduction with data retention," in *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*. New York, NY, USA: ACM, 2002, pp. 714–719.
23. A. Ferre and J. Figueras, "Characterization of leakage power in cmos technologies," in *Electronics, Circuits and Systems, 1998 IEEE International Conference on*, vol. 2, 1998, pp. 185 –188 vol.2.
24. A. Sinha and A. P. Chandrakasan, "Jouletrack: a web based tool for software energy profiling," in *Proceedings of the 38th annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 220–225.
25. K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Archit. Code Optim.*, vol. 1, no. 1, pp. 94–125, Mar. 2004.
26. J. Pallister, S. Hollis, and J. Bennett, "The impact of different compiler options on energy consumption," in *First LPGPU Workshop on Power-Efficient GPU and Many-core Computing*, ser. PEGPUM '13. New York, NY, USA: ACM, 2013.
27. F. Bellosa, "The benefits of event: driven energy accounting in power-sensitive systems," in *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, ser. EW 9. New York, NY, USA: ACM, 2000, pp. 37–42.
28. N. Chang, K. Kim, and H. G. Lee, "Cycle-accurate energy consumption measurement and analysis: case study of ARM7TDMI," in *Proceedings of the 2000 international symposium on Low power electronics and design*, ser. ISLPED '00. New York, NY, USA: ACM, 2000, pp. 185–190.
29. X. Fan, C. S. Ellis, and A. R. Lebeck, "The synergy between power-aware memory systems and processor voltage scaling," in *Proceedings of the Third international conference on Power - Aware Computer Systems*. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 164–179.
30. E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, ser. HotPower'10, Berkeley, CA, USA, 2010, pp. 1–8.
31. G. Hager, J. Treibig, J. Habich, and G. Wellein, "Exploring performance and power properties of modern multicore chips via simple machine models," *CoRR*, vol. abs/1208.2908, 2012.
32. V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, pp. 835–848, Jun. 2007.
33. T. Simunic, L. Benini, and G. De Micheli, "Cycle-accurate simulation of energy consumption in embedded systems," in *Design Automation Conference, 1999. Proceedings. 36th*, 1999, pp. 867 –872.
34. B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner, "Theoretical and practical limits of dynamic voltage scaling," in *Proceedings of the 41st annual Design Automation Conference*, ser. DAC '04. New York, NY, USA: ACM, 2004, pp. 868–873.
35. B. Calhoun, A. Wang, N. Verma, and A. Chandrakasan, "Sub-threshold design: The challenges of minimizing circuit energy," in *Low Power Electronics and Design, 2006. Proceedings of the International Symposium on*, 2006, pp. 366–368.

Table 3. Execution time fitting errors (%) of Equation 10 on the measured data given different CPU frequencies (f) and input size (2^N) at a 37°C core temperature.

	frequency f (GHz)														
	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6
$N = 6$	0.67	-1.90	-0.19	0.48	-0.27	-0.68	-1.11	-1.83	4.80	4.68	1.49	2.42	-6.14	0.82	-2.70
$N = 8$	1.26	-3.05	-1.49	-0.30	-0.94	5.71	-1.60	-1.97	4.15	4.28	1.02	1.95	-6.03	0.52	-2.84
$N = 10$	0.53	-1.08	-1.02	-0.40	-1.47	4.49	-2.24	-2.62	3.98	4.59	1.40	1.06	-6.03	1.50	-2.09
$N = 12$	1.58	-2.06	-4.08	-2.89	-0.42	3.20	2.34	0.72	6.39	7.06	3.77	-4.65	-4.28	-1.24	-4.39
$N = 14$	1.54	-4.00	-1.20	-0.61	-0.17	5.44	-0.72	-1.72	4.13	4.44	1.08	0.31	-6.25	0.76	-2.34
$N = 16$	1.49	-4.21	-1.31	0.41	-0.09	7.12	-0.41	-2.38	4.83	-2.97	2.40	1.57	-6.48	2.16	-1.30
|**average**|	1.18	2.71	1.55	0.55	0.56	4.21	0.62	1.63	4.71	3.68	1.86	0.44	5.87	0.75	2.6

Table 4. Power fitting errors (%) of Equation 13 on the measured data given different CPU frequencies frequencies (f) and input size (2^N) at a 37°C core temperature.

	frequency f (GHz)														
	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6
$N = 6$	3.00	0.94	-0.01	-1.05	-1.31	-1.33	-1.04	-0.52	0.29	0.49	0.46	0.86	0.26	-7.52	-0.60
$N = 8$	2.89	0.96	0.02	-1.06	-1.26	-1.35	-0.99	-0.44	0.24	0.50	0.36	0.87	0.29	-7.39	-0.60
$N = 10$	2.60	0.69	0.27	-0.80	-1.13	-1.22	-1.01	-0.32	-0.34	0.70	0.34	0.93	0.31	-7.52	-0.61
$N = 12$	2.43	0.52	0.05	-0.86	-1.07	-1.31	-0.90	0.15	-0.16	0.76	0.85	0.68	-0.59	-7.16	-0.26
$N = 14$	3.32	1.43	0.48	-1.07	-1.52	-1.72	-1.87	-0.81	-0.03	0.73	1.59	0.80	-0.01	-7.23	-0.71
$N = 16$	3.40	1.46	0.42	-1.11	-1.60	-1.85	-1.61	-0.98	-0.13	1.03	1.57	0.77	-0.09	-7.60	-0.69
|**average**|	2.94	1.00	0.20	0.99	1.31	1.46	1.24	0.49	0.02	0.70	0.86	0.82	0.03	7.40	0.58

Table 5. Energy fitting errors (%) of Equation 15 on the measured data given different CPU frequencies frequencies (f) and input size (2^N) at a 37°C core temperature.

	frequency f (GHz)														
	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6
$N = 6$	15.11	1.06	-0.27	-2.20	-3.23	-3.49	-3.31	-3.16	4.55	4.56	1.17	2.48	-6.92	-9.35	-4.72
$N = 8$	22.92	2.48	-0.05	-1.98	-2.99	3.41	-3.08	-2.58	4.47	4.82	1.20	2.67	-6.17	-8.87	-4.20
$N = 10$	8.63	0.46	-0.37	-2.50	-4.18	1.43	-4.83	-4.25	1.94	3.77	-0.21	-0.06	-8.05	-10.13	-5.70
$N = 12$	16.56	-1.90	-5.50	-6.07	-3.31	0.51	1.14	1.82	7.54	10.08	7.19	-1.38	-2.19	-6.58	-1.24
$N = 14$	22.89	1.22	0.26	-3.02	-3.12	2.35	-3.54	-2.56	4.57	6.05	3.80	2.04	-5.50	-6.87	-2.22
$N = 16$	24.23	1.69	0.41	-1.82	-3.03	3.82	-2.78	-3.51	5.04	-1.22	4.92	3.02	-6.11	-6.33	-1.53
|**average**|	18.39	0.83	0.92	2.93	3.31	1.34	2.73	2.37	4.69	4.68	3.01	1.46	5.83	8.02	3.27

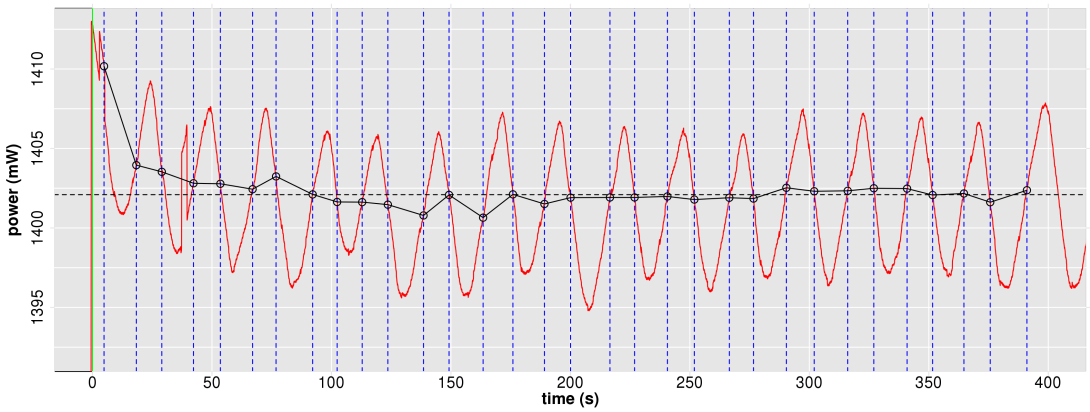


Fig. 9. Power trace for a CPU frequency of 1.0 GHz and a benchmark input size of 2^8 elements.

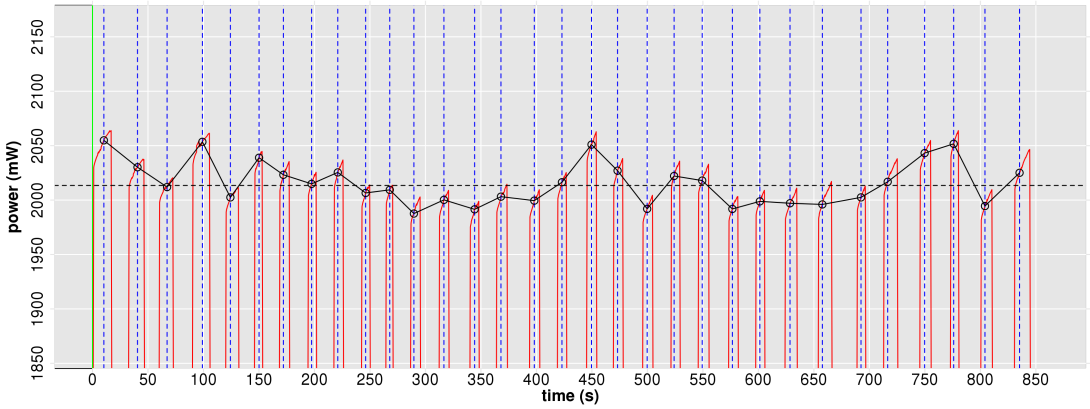


Fig. 10. Power trace for a CPU frequency of 1.2 GHz and a benchmark input size of 2^{12} elements.

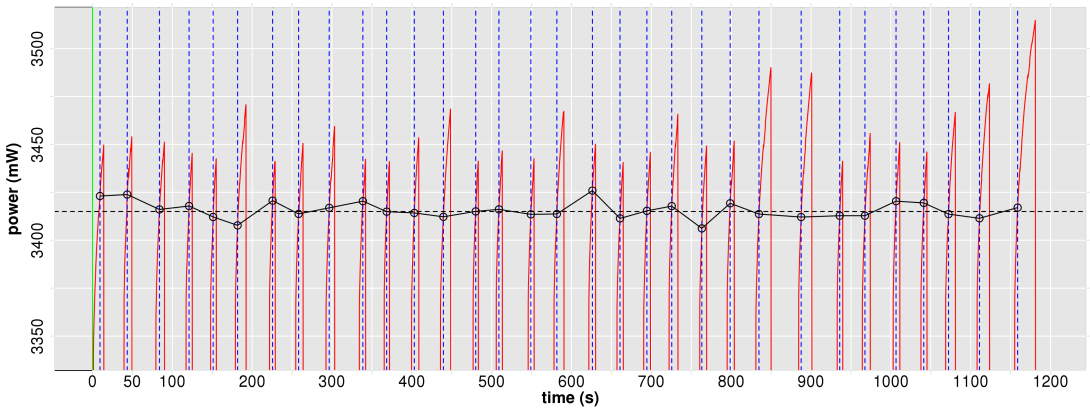


Fig. 11. Power trace for a CPU frequency of 1.5 GHz and a benchmark input size of 2^{14} elements.

Dépôt légal : 2013 - 4e trimestre
Imprimé à Télécom ParisTech – Paris
ISSN 0751-1345 ENST D (Paris) (France 1983-9999)

Télécom ParisTech

Institut Mines-Télécom - membre de ParisTech

46, rue Barrault - 75634 Paris Cedex 13 - Tél. + 33 (0)1 45 81 77 77 - www.telecom-paristech.fr

Département INFRES