The Energy/Frequency Convexity Rule: 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 provides both 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 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 consumption · 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 [9]. 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 users expect right now communication devices to provide data access every time, everywhere to everyone. 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 [2]. 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 [15]. For example, Aaron and Carroll [2] 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.

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 Sect. 6. We conclude and discuss future work in Sect. 7.

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: $P_{\text{total}} = P_{\text{system}} + P_{\text{CPU}}$, where P_{CPU} is consumed by the CPU itself and P_{system} by the rest of system. In a battery-powered hand-held computer device P_{system} may include the power needed to light the LCD display, to enable and maintain I/O devices (including memory), to keep sensors online (GPS, gyro-sensors etc.), and others.

The power consumption P_{CPU} of the CPU we focus on here can be divided into two parts: $P_{\text{CPU}} = P_{\text{dynamic}} + P_{\text{leak}}$, where P_{dynamic} is the power consumed by the CPU during the switching activities of transistors during 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: $P_{\text{dynamic}} = P_{\text{short}} + P_{\text{charge}}$. In the literature P_{charge} is usually [17] defined as $\alpha\, C f V^2$, 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 and V the voltage swing across C.

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. Quantifying P_{short} is gate specific but it may be approximated 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 $P_{\text{charge}} + (\eta - 1) P_{\text{charge}}$, i.e., $\eta \cdot \alpha C_L f V^2$, 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. Some leakage currents are induced during the *on* or *off*-state of the transistor, or both. Six distinct sources of leakage are identified [12]. 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 [1]. Leakage current models, e.g., as incorporated in the BSIM [12] micro models, are accurate yet 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 portal power can be rewritten as $P_{\text{total}} = P_{\text{system}} + P_{\text{leak}} + P_{\text{dynamic}}$.

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

$$E(\Delta t) = \int_0^{\Delta t} P(t)\ dt = \int_0^{\Delta t} I(t) \cdot V(t)\ dt, \qquad (1)$$

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*.

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 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 kernel was patched to print a temperature sample to the kernel debug output at a rate of 2 Hz.

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, 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 s 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.

Data was fitted using `R` and the `nls()` function employing the *port* algorithm.

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, we need to take account for it 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, processing kernel events, managing memory etc. When the processor is not spending time in kernel mode, 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 needs a fixed amount of clock cycles cc_{k} per time unit to complete its tasks. Thus, we propose to model the amount of clock cycles to complete a benchmark sequence of instructions cc_{b} as $t(f^{\beta} - cc_{\mathrm{k}})$, where cc_{k} are the number of clock cycles spent in the OS, 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:

$$t = \frac{cc_{\mathrm{b}}}{f^{\beta} - cc_{\mathrm{k}}}. \tag{2}$$

Note that t tends to zero for $f \to \infty$ and there is a vertical asymptote at $\sqrt[\beta]{cc_{\mathrm{k}}}$.

Table 1 shows the fitting errors of Eq. 2 on the execution time measurement data, averaged over all tested input sizes. The fitting exhibits a vertical asymptote around 115 MHz. This may indicate the minimum amount of clock cycles required by the OS of the phone to operate. The measurement data for input sizes 2^6 up to 2^{16} are well described by Eq. 2. 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 Sect. 2), the case for leakage is more involved, and warrant a longer presentation. In particular, leakage power is

Table 1. Average absolute execution time (t), power (P), and energy (E) fitting errors (%) of Eq. 2, 4 and 5 respectively, on the measured data given different CPU frequencies (f) at a 37 °C core temperature.

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
error t	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	2.61
error P	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
error E	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

heavily temperature-dependent [12]. 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. [18] shows similar results for a 0.1 µm 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. The Berkeley Short-channel IGFET Model (BSIM) [12] shows that the leakage current micro models depend on a multitude of variables. The temperature itself appears several times in the sub-threshold and BTBT leakage models; the gate leakage however is not temperature dependent. Mukhopadhyay et al. [13] 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 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.

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) [5, 11, 16]. After analysis on our data, we discovered that none of the cited approximations would fit well. This is because the rationale 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.

Skadron et al. [14] 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):

$$R_T = \frac{P_{\text{leak}}}{P_{\text{dynamic}}} = \frac{R_0}{V_0 T_0^2} e^{\frac{B}{T_0}} V T^2 e^{\frac{-B}{T}}. \tag{3}$$

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 γ, which includes the temperature dependent variables and other constants. Total power P_{total} is thus:

$$\begin{aligned} P_{\text{total}} &= P_{\text{system}} + P_{\text{leak}} + P_{\text{dynamic}} \\ &= P_{\text{system}} + \gamma V P_{\text{dynamic}} + P_{\text{dynamic}} \\ &= P_{\text{system}} + (1 + \gamma V) \cdot \eta \alpha C f V^2. \end{aligned} \tag{4}$$

This formulation of P_{total} incorporates three parameters: P_{system}, γ, and $\eta\alpha C$. The values of these variables can be obtained via fitting power traces on Eq. 4. V and f are linked via the DVFS process inherent to the Linux kernel and the hardware technicalities. Experimental values for our CPU are found inside the Siyah kernel; they are shown in Table 2. The power fitting errors are shown in

Table 2. Frequency and voltage relationship for the Dynamic Voltage and Frequency Scaling (DVFS) process in the default Siyah kernel.

f (MHz)	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600
V (mV)	920	950	950	950	975	1000	1025	1075	1125	1175	1225	1250	1275	1325	1350

Table 1 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 Eq. 4.

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 Eq. 1 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 Eq. 2, and power in Eq. 4, can be used to model the energy of one benchmark kernel. The energy consumed by the CPU E_{CPU} is given by

$$\begin{aligned} E_{\text{CPU}} &= E_{\text{leak}} + E_{\text{dynamic}} \\ &= P_{\text{bench}} \cdot t_{\text{bench}} \\ &= \big((1+\gamma V)\cdot \eta\alpha C f V^2\big) \cdot \frac{cc_{\text{b}}}{f^{\beta} - cc_{\text{k}}}. \end{aligned} \tag{5}$$

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

5 The Energy/Frequency Convexity Rule

Using the testbed and models described above, Fig. 1 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 1 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 1). 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.

Figure 1 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

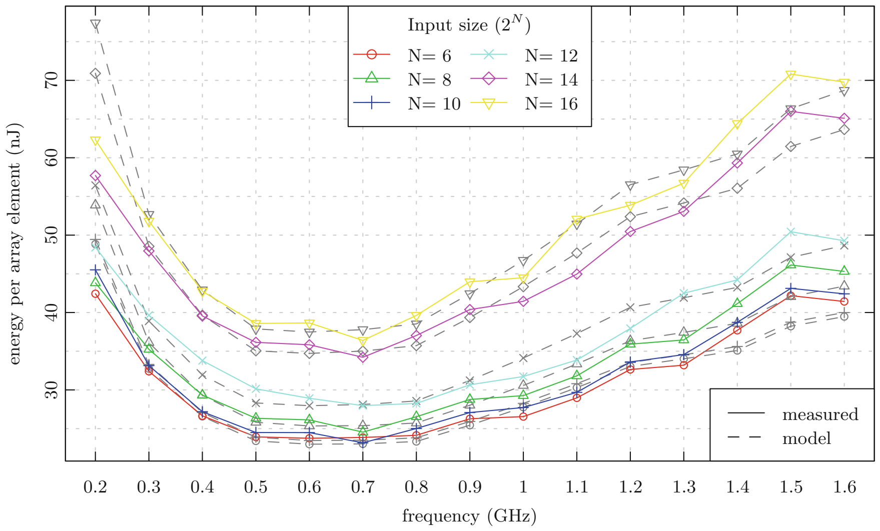

Fig. 1. Energy required by the CPU at 37 °C to complete our benchmark kernel given an input size. The dashed lines denote the theoretical curve as per Eq. 5.

consumption of the benchmark kernel scales approximately linearly with the number of instructions. The time Δt it takes to execute an instruction sequence increases more than linearly with decreasing operating frequency. 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 2 between the *frequency* (GHz) and *voltage* (V) in the Linux kernel with a linear approximation: $V = m_1 f + m_2$. Now the derivative of E_{CPU} defined in Eq. 5 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.

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

$$\frac{(1+\gamma V)V f^{\beta}\beta}{f^{\beta} - cc_{\text{k}}} = f m_1(3\gamma V + 2) + (1+\gamma V)V. \qquad (6)$$

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, $E_{\mathrm{CPU,min}}$ will shift to a higher frequency, and, if m_1 or γ decreases, f_{opt} will decrease as well. γ is temperature dependent; if the temperature increases, γ will increase accordingly. Hence, $E_{\mathrm{CPU,min}}$ and f_{opt} increase with temperature as well. For the presented measurements Eq. 5 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 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 [4,10,15]. The authors put forward some motivation, but do not provide an analytical framework. Other studies, e.g., Hager et al. [8] and Freeh et al. [6], discuss what the consequences are of said behavior and how to exploit them, from a high-level point of view. A detailed explanation or an analytical derivation, as presented here, is however not highlighted.

7 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 parameterized 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. A more in depth analysis of the Energy/Frequency Convexity Rule can be found in our technical report [3].

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.

References

1. Agarwal, A., Mukhopadhyay, S., Kim, C., Raychowdhury, A., Roy, K.: Leakage power analysis and reduction: models, estimation and tools. IEEE Proc. Comput. Digital Tech. **152**(3), 353–368 (2005)
2. Carroll, A., Heiser, G.: An analysis of power consumption in a smartphone. In: Proceedings of the USENIX Conference on USENIX, Berkeley (2010)
3. De Vogeleer, K., Memmi, G., Jouvelot, P., Coelho, F.: Energy consumption modeling and experimental validation on mobile devices. Technical Report 2013D008, TELECOM ParisTech, December 2013
4. Fan, X., Ellis, C.S., Lebeck, A.R.: The synergy between power-aware memory systems and processor voltage scaling. In: Falsafi, B., VijayKumar, T.N. (eds.) PACS 2003. LNCS, vol. 3164, pp. 164–179. Springer, Heidelberg (2005)
5. Ferre, A., Figueras, J.: Characterization of leakage power in cmos technologies. In: 1998 IEEE International Conference on Electronics, Circuits and Systems, vol. 2, pp. 185–188 (1998)
6. Freeh, V.W., Lowenthal, D.K., Pan, F., Kappiah, N., Springer, R., Rountree, B.L., Femal, M.E.: Analyzing the energy-time trade-off in high-performance computing applications. IEEE Trans. Parallel Distrib. Syst. **18**(6), 835–848 (2007)
7. Gold, B., Rader, C.M.: Digital Processing of Signals. McGraw-Hill, New York (1969)
8. Hager, G., Treibig, J., Habich, J., Wellein, G.: Exploring performance and power properties of modern multicore chips via simple machine models. CoRR abs/1208.2908 (2012)
9. Ickin, S., Wac, K., Fiedler, M., Janowski, L., Hong, J.H., Dey, A.: Factors influencing quality of experience of commonly used mobile applications. IEEE Commun. Mag. **50**(4), 48–56 (2012)
10. Le Sueur, E., Heiser, G.: Dynamic voltage and frequency scaling: the laws of diminishing returns. In: Proceedings of the 2010 International Conference on Power Aware Computing and Systems, HotPower'10, Berkeley, pp. 1–8 (2010)
11. Liao, W., He, L., Lepak, K.M.: Temperature and supply voltage aware performance and power modeling at microarchitecture level. Trans. Comp. Aided Des. Integ. Cir. Sys. **24**(7), 1042–1053 (2006)
12. Liu, W., Jin, X., Kao, K., Hu, C.: BSIM 4.1.0 MOSFET model-user's manual. Technical Report UCB/ERL M00/48, EECS Department, University of California, Berkeley (2000)
13. Mukhopadhyay, S., Raychowdhury, A., Roy, K.: Accurate estimation of total leakage current in scaled cmos logic circuits based on compact current modeling. In: Proceedings of the Design Automation Conference 2003, pp. 169–174, June 2003
14. Skadron, K., Stan, M.R., Sankaranarayanan, K., Huang, W., Velusamy, S., Tarjan, D.: Temperature-aware microarchitecture: modeling and implementation. ACM Trans. Archit. Code Optim. **1**(1), 94–125 (2004)
15. Snowdon, D.C., Ruocco, S., Heiser, G.: Power management and dynamic voltage scaling: Myths and facts. In: 2005 WS Power Aware Real-time Comput. New Jersey, September 2005
16. Su, H., Liu, F., Devgan, A., Acar, E., Nassif, S.: Full chip leakage estimation considering power supply and temperature variations. In: Proceedings of the 2003 International Symposium on Low power Electronics and Design, ISLPED '03, pp. 78–83. ACM, New York (2003)

17. Weste, N.H.E., Eshraghian, K.: Principles of CMOS VLSI Design: A Systems Perspective. Addison-Wesley Longman Publishing Co. Inc., Boston (1985)
18. You, Y.-P., Lee, Ch., Lee, J.-K.: Compiler analysis and supports for leakage power reduction on microprocessors. In: Pugh, B., Tseng, C.-W. (eds.) LCPC 2002. LNCS, vol. 2481, pp. 45–60. Springer, Heidelberg (2005)