DOI 10.15622/ia.22.5.3

M. PELOGEIKO, S. SARTASOV, O. GRANICHIN

ON STOCHASTIC OPTIMIZATION FOR SMARTPHONE CPU ENERGY CONSUMPTION DECREASE

Pelogeiko M., Sartasov S., Granichin O. **On Stochastic Optimization for Smartphone CPU Energy Consumption Decrease.**

Abstract. Extending smartphone working time is an ongoing endeavour becoming more and more important with each passing year. It could be achieved by more advanced hardware or by introducing energy-aware practices to software, and the latter is a more accessible approach. As the CPU is one of the most power-hungry smartphone devices, Dynamic Voltage Frequency Scaling (DVFS) is a technique to adjust CPU frequency to the current computational needs, and different algorithms were already developed, both energy-aware and energy-agnostic kinds. Following our previous work on the subject, we propose a novel DVFS approach to use simultaneous perturbation stochastic approximation (SPSA) with two noisy observations for tracking the optimal frequency and implementing several algorithms based on it. Moreover, we also address an issue of hardware lag between a signal for the CPU to change frequency and its actual update. As Android OS could use a default task scheduler or an energy-aware one, which is capable of taking advantage of heterogeneous mobile CPU architectures such as ARM big.LITTLE, we also explore an integration scheme between the proposed algorithms and OS schedulers. A model-based testing methodology to compare the developed algorithms against existing ones is presented, and a test suite reflecting real-world use case scenarios is outlined. Our experiments show that the SPSA-based algorithm works well with EAS with a simplified integration scheme, showing CPU performance comparable to other energy-aware DVFS algorithms and a decreased energy consumption.

Keywords: Android OS, dynamic voltage frequency scaling, stochastic optimization, SPSA, energy consumption.

1. Introduction. Mobile devices, such as smartphones, tablets, and smartwatches, became an integral part of modern life. Digital services provided by these devices both improve quality of life and form new ways of social interactions. More than 6.3 billion smartphone subscriptions were active, and in 2024 this figure is expected to exceed 7 billion people. The most common operating system (OS) for mobile devices today is Android [1].

As the battery capacity of mobile devices is limited, power consumption optimization becomes an increasingly important task – no one wants to have an important call interrupted because of a discharged battery. This issue could be alleviated by applied software, which is capable of providing energy consumption models and eco-friendly profiles. Then, a boost of device longevity is provided by new hardware technologies like Li-Pol batteries or heterogeneous big.LITTLE CPU architecture. Finally, Android OS contains a power management subsystem which includes three components: energy-aware scheduling (EAS), dynamic voltage frequency scaling (DVFS) governors and idle state (IS) governors, so improving underlying algorithms improves energy consumption as well.

In essence, the DVFS algorithm analyzes the current state of the smartphone and advises the CPU to switch to a particular frequency, so that the algorithm could meet some optimization criteria. Performance and energy consumption cannot be optimized at the same time, because to save energy one has to set a low frequency, and higher computational capabilities call for higher battery resources. Therefore, every energy-aware DVFS algorithm finds some form of balance by introducing combined optimization criteria. Ideally, applications run as slow as perceivable comfortable to save energy.

A significant part of research is mainly concentrated on DVFS algorithms or better energy models for task scheduling, and there is a reason behind such attention – energy-aware task allocation combined with online CPU performance control could result in considerable energy savings. Simultaneous perturbation stochastic approximation (SPSA) is one of the possible ways to track optimal CPU frequency in terms of both energy consumption and performance. On average, the algorithm will nearly follow the steepest descent direction [2]. SPSA could be considered a random search technique, and it helped in solving various computer science-related tasks [3]. In our previous work, we investigated an SPSA-based DVFS governor using a single noisy observation [4], and while working on par with commonly used algorithms, further experiments confirmed that an approach proposed there, although already usable, was not final. SPSA with two noisy observations is more stable compared to one observation version [5]. Thus, the optimal frequency tracking process converges faster on average.

The main contribution of this paper is a proof of the concept that the energy-aware DVFS algorithm could be built based on SPSA with two noisy observations. Additionally, we formulate a different average risk function which takes into account the *cost of execution* of a program at a particular frequency to save energy. Technical considerations are also investigated, for example, whether is it worth running our new DVFS governor with EAS or Completely Fair Scheduler (CFS), and how to connect it with EAS when needed. We also take into account the notable fact that the CPU frequency change cycle could be considerably longer than the DVFS cycle.

This paper is organized as follows. In Section 2, an overview of the DVFS and EAS alongside the existing algorithms for modern smartphones for those subsystems is given. It also contains a brief overview of stochastic optimization in general, a description of the SPSA approach and a review of the previous work in the field. Section 3 contains a description of the proposed DVFS algorithms. Experimental setup and methodology discussion are presented in Section 4, while experimental results are analyzed in Section 5. Final remarks are given in Section 6 to conclude this paper.

2. Overview

2.1. DVFS and EAS. CPUs in modern smartphone systems-on-a-chip are always multicore. Generally, this architecture is homogeneous – each core has the same computational capabilities and power profile. However, there is a novel approach for mobile device CPU design reflected in the ARM bid.LITTLE architecture – a heterogeneous CPU. Cores are divided into several clusters, and each core is homogeneous only within its cluster. Therefore it is possible to run computationally non-demanding programs on weaker, but energy efficient (so-called "LITTLE") cores, while more powerful and power-hungry ("big") cores are utilized for prioritized or computationally intensive tasks. A real-world example of such a design is given in Table 1.

Table 1. Xiaomi Redmi Note 8 Pro CPU clusters frequencies

A55			A76		
F (Hz)	I (mA)	I/F (mA/GHz)	F (Hz)	I (mA)	I/F (mA/GHz)
2000000	90.04	45.02	2050000	324.33	158.21
1933000	85.8	44.39	1986000	307.98	155.08
1866000	80.27	43.02	1923000	291.52	151.60
1800000	72.77	40.43	1860000	269.61	144.95
1733000	66.61	38.44	1796000	247.53	137.82
1666000	62.05	37.24	1733000	233.56	134.77
1618000	58.95	36.43	1670000	209.73	125.59
1500000	52.33	34.89	1530000	177.39	115.94
1375000	44.83	32.60	1419000	152.46	107.44
1275000	39.69	31.13	1308000	130.33	99.64
1175000	35.5	30.21	1169000	105.19	89.98
1075000	31.24	29.06	1085000	91.11	83.97
975000	27.86	28.574	1002000	79.53	79.37
875000	25	**28.571**	919000	70.65	76.88
774000	23.5	**30.36**	835000	61.38	73.51
500000	19.55	**39.10**	774000	56.85	73.45

It is important to note for our research that core configuration is available within OS, and operating frequency is set uniformly for the entire cluster, that is, every core within a cluster always works at the same frequency. Different clusters might work at different frequencies at the same time.

DVFS as a technique is a control over CPU operating frequency and voltage at runtime to increase or decrease CPU performance at a cost of power consumption. Generally, CPU power consumption is related to the cube of its frequency, and as a side effect CPU heats. However, cooling mechanisms available to mobile devices are limited both by power usage and form factor, therefore most of the time heat could only be dissipated by surrounding air. Thus therefore, operating frequencies are limited to those producing an amount of heat that could be consistently dissipated – at the time of writing, about 2 GHz for commercially available CPUs.

Although dependency between power consumption and frequency is still vaguely close to linear at these levels, power efficiency – the relation of power to performance – is clearly exponential [6]. Power efficiency could also be interpreted as the cost of instruction execution at a selected frequency. A complimentary optimization criterion is the execution time, so in practice, the goal is to achieve a certain balance between power and performance. Android OS DVFS governors are OS modules that observe the current state of a device and send signals to the CPU to increase or decrease operating frequency. For example, `powersave` and `performance` governors set minimum and maximum frequency respectively.

Although we say that DVFS governors "set" some frequency to a cluster, it is technically a recommendation to the CPU, not a direct assignment. Frequency change does not happen immediately, and DVFS cycle length – the time between governor invocations – could be considerably lower than the frequency change time. For example, the default cycle length for the `OnDemand` governor is 10 ms, while both clusters of Xiaomi Redmi Note 8 Pro CPU take about 30 ms to change frequency. So while the first governor invocation could initiate hardware frequency update, from the CPU perspective the next two invocations are as good as non-existing.

Energy-aware scheduling (EAS) [8] is an Android OS task scheduler, which could only operate in heterogeneous CPU topologies such as ARM big.LITTLE. Power management for symmetric topologies is uniform, that is, CPU frequency is set globally for the entire CPU, therefore, the maximum occupancy and performance are the same for every core. EAS uses a normalized energy model for every core cluster, taking into account available frequencies and power usage. When a task needs to be scheduled, EAS chooses the core to run this task in such a way that CPU power consumption will increase in the least possible way. However, EAS works while the CPU load is lower than 80%, otherwise, a Completely Fair Scheduler is used until the load is decreased. In order to obtain consistent power savings, EAS should be able to issue signals to hardware to control the CPU clusters' maximum occupancy, and `schedutil` DVFS governor is considered to be a part of EAS.

2.2. DVFS algorithms. There is a number of commonly available DVFS governors for Android OS. Some of them are Android-specific, while others were initially used in the Linux kernel [9]:

PowerSave: This governor sets the CPU to the minimal available frequency and keeps it forever. This governor saves the most energy but provides the worst performance.

Performance: This governor works exactly the opposite, it sets the maximum frequency for the best performance and highest energy consumption. `Performance` and `PowerSave` reflect two extreme optimization strategies.

OnDemand: This governor sets cluster frequency proportionally to the maximum core load – active time divided by total time – within a cluster observed between governor invocations, so the device remains responsive. When a particular CPU load threshold ($\sim 80\%$) is reached, the maximum frequency is set until the load is again below the threshold.

Conservative: An improvement over `OnDemand`, this governor gradually increases the frequency when there is activity on the CPU and decreases it is to the lowest value when there is none to little activity.

Interactive: This governor is developed specifically for Android OS with UI interaction in mind. Operating frequency is once again dependent on the activity level, but activity level evaluation is event-driven in addition to timer-driven. User interaction such as screen touch is among tracked events.

schedule: This is a governor designed to work exclusively with EAS. Its basic idea is the same as in `OnDemand`, but the definition of the load is based on the EAS energy model instead of the active time to total time ratio.

While there is a large number of DVFS algorithms created by standalone developers to enhance the characteristics of the default algorithms, additional approaches are covered in the literature.

Research is done on replacing EAS and `schedutil` with a different approach for better energy efficiency. Unlike the static energy model of EAS, AdaMD [10] implements a scheduling routine which regularly inspects various resources that are currently required by executed processes and reassigns them to a more suitable cores if needed. Compared to other thread-to-core approaches, this technique allows to save up to 28% o energy while satisfying 95% of performance constraints.

In recent years there is a research trend to evaluate neural network usage for general-purpose DVFS governors. For example, the "Long Short-term Memory" effect of the recursive network could be used, as correlations should be made only for short-term data to prevent gradient attenuation problems [11].

Such network architecture reduces CPU power consumption by a maximum of 19% in comparison to common DVFS.

DVFS could be built in mind with other criteria than performance or energy saving. Chip temperature is another important factor, and by using decisions from the relatively simple neural network in the DVFS process average chip temperature could be reduced by up to 18 $\deg$ C with a minimum execution overhead and comparable performance [12]. However, chip temperature is not mutually exclusive with energy efficiency, as using deep reinforcement learning neural network to determine the temperature of the chip and estimating environmental temperature could result in 23.9% less energy consumption while retaining the required level of performance [13].

Application-specific DVFS models could also be built. For instance, a DVFS model for augmented reality applications that takes into account the requested frame rates and response time could theoretically be decreased by up to 80%, but it was not proved in practice yet [14].

Graphical Processing Unit (GPU) is a powerful device to offload computations from the CPU, and it also could operate on different frequencies, so it is reasonable to make a joint DVFS strategy for both devices. A simple rule-based model could reduce energy consumption by 18.11%, while the smartphone frame rate drops by 3.12% [15]. Another approach is to aggregate load, energy and temperature data for CPU, GPU and RAM, and assign frequency for each component by joint priorities list. This technique saves at least 26.8% power compared to default governors and state-of-the-art approaches [16].

2.3. Idle states and idle state governors. Modern CPUs are capable to enter idle states where program execution is suspended. Multi-core processors can set one or more of their cores to idle state, while other cores remain active. While in an idle state, part of the processor hardware is switched off, so it consumes less power. The deeper the state, the more hardware is switched off, but at the cost of greater entry and exit times. In Linux and Android OS terms, the total enter latency and minimum time hardware would stay in a particular idle state is called target residency [17].

For example, without going into much detail, here is a list of idle states of dual-core ARM CortexA9 processor [18] in order of increasing idle depth:

– C0 – active state.

– C1 (WFI) – most CPU timers are deactivated. Exit latency is 4 us.

– C2 ((CPUs OFF, MPU + CORE INA) – CPU is off, memory protection unit (MPU) is activated to protect critical data, and the core is inactive. Exit latency is 1100 us.

– C3 (CPUs OFF, MPU + CORE Closed Switched with Retention) – similar to C2, but the core is in CSWR mode. Exit latency is 1200 us.

– C4 (CPUs OFF, MPU CSWR + CORE Open Switched Retention) – similar to C3, but the core is in OSWR mode. Exit latency is 1500 us.

It is important to note that ability of a CPU to enter a particular state could be or could be not utilized by mobile device system-on-chip and OS, therefore, it is possible to observe a smartphone capable of entering only C1, while its CPU could go deeper by design.

The idle state governor is a module within the Android OS kernel that can track the current system state and send a signal to the CPU to enter or exit some idle state for one or more of its cores. There are several default algorithms generally available in Android devices, and at the time of writing the default algorithm is the `menu`. It tries to predict the current idle duration, then adjusts the obtained value based on a number of factors, and then tries to find the deepest idle state given its target residency and exit latency. The prediction of the current idle duration is based on the history of the previous idle times.

2.4. Evaluating Energy Consumption. Measuring smartphone CPU energy consumption is a non-trivial task, and there are several approaches available. We classify them into direct and non-direct approaches [7].

Direct approaches involve the physical measurement of momentary CPU electrical parameters either with external or internal sensors. Energy consumed over a period of time could be estimated as:

$$E = \int_0^t U(t)I(t)\,dt,$$

where $U(t)$ is momentary voltage, $I(t)$ is momentary current. While it is technically possible to have separate circuits to power the CPU at different voltages, it is challenging, so in commercially available smartphones DVFS is in fact dynamic frequency-only scaling, while voltage changes only due to natural processes within a battery[1]. Therefore, we may rewrite the above formula as:

$$E = U\int_0^t I(t)\,dt, \tag{1}$$

[1]Battery voltage drops during discharge, but between 100% and 20% the change is technically negligible by electronic components

where U is a nominal voltage on a CPU. Equation (1) allows to simplify measurement scheme and use only an ammeter. In practice, as ammeter readings are discrete, the Riemann sum is calculated instead of the actual integral.

While this approach could provide the most reliable results, there are considerable limitations to it. First of all, while the internal sensors are rather common for smartphone peripherals such as Bluetooth and Wi-Fi modules, they are seldom used to estimate the CPU power consumption [2]. The frequency and momentary current multiple times over a single second. An external ammeter should be able to catch such high-frequency changes, so most advanced commercially advanced ammeters operate in KHz range [19], but their cost could be a prohibitive factor. Additionally, ammeters should be connected, consequentially, with the measured device. Smartphone electronics do not allow us to easily connect CPU power input with ammeter. Alternatively, one could connect an ammeter to a battery connection slot while also using an external power device at a constant voltage, but readings acquired with this approach will be inevitably skewed by other smartphone peripherals such as a screen or Wi-Fi module.

Non-direct approaches estimate energy consumption by associating statistics unrelated to energy to it by some model. For example, one could estimate how much energy is consumed by a specific CPU instruction [20]. Due to the size of instruction sets in modern CPUs, we consider this approach impractical and instead follow the model proposed by Google [21]. Under this model, a CPU operating at a specified frequency consumes a specific constant current. Therefore, equation (1) is simplified even further as:

$$E = U \sum_{i=1}^{n} I(f_i) t_{fi}, \tag{2}$$

where n – number of frequencies available to CPU, f_i – particular operating frequency, $I(f)i)$ – constant current at a specified frequency, t_{fi} – time spent by CPU at a frequency f_i.

Equation (2) is further supported by Android OS, as t_{fi} is stored in special `time-in-state` files in the `/sys` directory. Temporal data is stored in separate lines for each frequency as “`<frequency><time>`”. The number of lines is equal to n. It is worth noting, that the CPU clusters under big.LITTLE architecture are treated as separate CPUs, so lines in `time-in-state` also

[2]Anecdotally, we didn’t encounter them in any devices available to us.

differ from core to core depending on the cluster they belong to. Time is measured in 10 milliseconds units, and the count is started when the corresponding OS driver is installed or reset to measure processor data. Values for $I(f_i)$ are stored in a weight coefficients file called `power_profile.xml`. This file should be provided by the smartphone manufacturer, but unfortunately, it is not always the case. However, absent power constants could be extracted from other smartphones using the same CPU model with the same or similar core topology. An example of such file contents is given in Table I.

To estimate power consumption one needs to multiply timing data to corresponding weight coefficients in a device `power_profile.xml` - a file provided by a smartphone manufacturer which contains power metrics for each smartphone device or peripheral. A sample of its contents related to CPU is shown in Table I. Power constants there are reported in mA[3].

Different DVFS governors produce different time distributions over available sets of frequencies. Methodologically, when a voltage is constant, electrical charge is a main indicator of energy consumption and Equation (2) could be rewritten as:

$$q_{fi} = I(f_i)t_{fi},$$

$$q = \sum_{i=1}^{n} q_i,$$

$$E = Uq,$$

where q_{fi} is an electrical charge spent at i-th frequency, q – total electrical charge. Because of this, we report energy consumption in our experiments in mAh.

2.5. Simultaneous Perturbation Stochastic Approximation. In a significant number of control problems target system behavior could be described in the form of empirical quality functional (also known as medium risk functional). An optimal control action is taken based on the extrema of this functional. In our case, CPU operating frequency might be determined based on the list of available frequencies, current CPU workload and its history and other quality criteria such as energy consumption.

[3]It is necessary to point out, that, in general, power constants reported in mA are not enough to make conclusions on energy consumption, as voltage would also be required. However, as constant nominal voltage for mobile CPUs available to market is 1V, current constants are numerically equal to power constants.

More formally, let $F_t(x, w)$ be a function of discrete time t, some parameter x and randomised vector w. The medium risk functional is defined as:

$$f_t(x) = E_w F_t(x, w),$$

and the minimum point of $f_t(x)$ as

$$\theta_t = \arg\min_x f_t(x).$$

Then in order to find the optimal point the task is to build the sequence of estimations $\{\hat{\theta}_n\}$ such that $||\hat{\theta}_n - \theta_t|| \to \min$ based on observations of the random variables $F_t(x_n, w_n)$, n = 1, 2,..

We define a *momentary trial perturbation* as a sequence of the observed uniformly symmetrically distributed independent random vectors Δ_n with covariance matrices:

$$cov\{\Delta_n \Delta_j^T\} = \delta_{nj}\sigma_\Delta^2 I,$$

where $\delta_{nj} \in \{0, 1\}$ is the Kronecker symbol, $0 < \sigma_\Delta < \infty$. The Bernoulli random vectors are suitable and frequently used as a simultaneous trial perturbation because the vector coordinates Δ_n are independent of one another and have equiprobable values of ± 1.

It is possible to use three following algorithms when observations are noisy:

$$\hat{\theta}_n = \hat{\theta}_{n-1} - \frac{\alpha_n}{\beta_n}\Delta_n y_n,$$

$$\hat{\theta}_n = \hat{\theta}_{n-1} - \frac{\alpha_n}{2\beta_n}\Delta_n(y_n^+ - y_n^-),$$

$$\hat{\theta}_n = \hat{\theta}_{n-1} - \frac{\alpha_n}{\beta_n}\Delta_n(y_n^+ - y_n),$$

to build a minimum point estimation sequence for a functional $F(x)$ without significant loss of convergence rate [22]. We denote noisy observations in the following way:

$$y_n = F(\hat{\theta}_{n-1}, w_n^+) + v_n,$$

$$y_n^- = F(\hat{\theta}_{n-1} - \beta_n \Delta_n, w_n^+) + v_n,$$

$$y_n^+ = F(\hat{\theta}_{n-1} + \beta_n \Delta_n, w_n^+) + v_n.$$

$\{\alpha_n\}$ and $\{\beta_n\}$ here are sequences of non-negative numbers conforming to a set of conditions, w_n^+ is a stochastic perturbation vector for y_n^+ observation, v_n^+ is an arbitrary external noise during the observation. This recurrent procedure is called *simultaneous perturbation stochastic approximation* (SPSA) because it inseparably contains a randomized trial perturbation which is simultaneous in all coordinates. Overall, SPSA could be classified as a stochastic gradient descent algorithm.

The first algorithm uses only a single noisy observation, and the second and third involve two noisy observations. For the purposes of distinction between those variations of the SPSA algorithm, we will call the version with a single noisy observation as SPSA1, and the latter two variations – SPSA2.

Among the conditions for consistency of estimates we specifically set out a condition for a weak correlation between the trial perturbation $\{\Delta_n\}$ and sequences of indeterminacies $\{w_n\}$ and $\{v_n\}$ as the most important. SPSA1 has lower mean squared convergence rate compared to SPSA2 but has the advantage of using only a single noisy observation, which could be more time-efficient at the end, hence both variations could be used in practice.

Both variations of the SPSA algorithm follow the same general flow:

1. Define an empirical functional $F(x)$.
2. Make an initial optimal estimate of $\hat{\theta}_0$.
3. Perturb a current optimal estimate.
4. Obtain the required amount of the noisy observations of F and update a current optimal estimate $\hat{\theta}_n$.
5. Go to Step 3.

Previously, we developed a DVFS governor based on SPSA with a single noisy observation [4]. It was shown, that overall its energy consumption is between `OnDemand` and `Interactive` governors under the CFS scheduler with a performance drop of no more than 3.06%, but in some specific scenarios it could save up to 16% of the total CPU energy.

3. Algorithm

3.1. Defining The empirical Functional. Empirical functional is a crucial component for SPSA algorithm. Although from a theoretical perspective, its definition could vary significantly even within the same problem [2], we used the lessons learned of the SPSA1 governor and added the following considerations to the functional definition.

We start our discussion by defining a *load* L on the CPU core as the percentage of active CPU time from the total CPU time (active and idle) over some period of time. This definition is the same as the one defined for the `OnDemand` governor. Then, a *computational volume* or simply *volume* denotes a product of frequency by load. Essentially, the volume is a number of the CPU ticks dedicated to active computational processes, and it is closely related to the number of the executed instructions. The crucial observation for DVFS is that the same volume produces different loads under different frequencies: the higher the frequency, the lower the load, and vice versa.

The task of DVFS is peculiar in the sense that we can base our DVFS strategy on the history of the CPU load observations among other criteria, but still, the future load cannot be reliably determined – for example, we can't predict that smartphone would receive a phone call in the next second. Therefore, we introduce *target load* or *threshold load* L_T – a specific constant value of load over some observation time. In our experiments we used empirically determined values of L_T from 60% to 80% – such values provide additional computational capacity in a scenario where the actual load proved to be much higher than anticipated from the load history. L_T is used as a boundary between two different DVFS strategies:

– If the current load exceeds L_T, we assume that the smartphone performs a rather long computational task. Therefore, all CPU resources should be made available for it, and the optimal frequency that DVFS should set is the one that provides the closest load to L_T under the currently observed volume. In this case, performance considerations outweigh energy savings. Long-term, if a load doesn't drop below L_T, the maximum frequency will eventually be set until the load drops.

– If the load does not exceed L_T, it means there is room to optimize energy consumption by finding such a frequency that provides the best energy efficiency and the projected load still does not exceed L_T.

We now define *execution efficiency* or *cost-of-execution (CoE)* of an operating frequency f in the following way:

$$CoE = \frac{E(t)}{n_{ticks}} = \frac{\int_0^t U(t)I(t)\,dt}{ft} = \frac{I_f U}{f},$$

where CoE is cost-of-execution, $E(t)$ is energy spent over a period of time, n_{ticks} is a number of CPU ticks observed during that time, t is the observed period of time. The discussion on translation from integral to constant product is given in Subsection 2.4. In essence, CoE is the energy cost of one CPU tick under a specific operating frequency.

CoE could also be viewed as a priority for the DVFS governor for the frequency selection – the lower the value, the higher the priority. Under this point of view, some *generalized cost-of-execution GCoE* metric could be used instead of CoE to prioritize the frequency selection, as it only needs to establish order relationship between frequencies.

In our research, we defined it as:

$$GCoE = \frac{I_f}{f}.$$

As discussed, the CPU voltage is safe to be treated as constant, so by removing U term from the CoE definition we do not change the established order relationship[4].

It is important to note that, given the power constants available, with the increase of f I_f also increases, but $GCoE(f)$ is generally a non-monotonically increasing function. An example is given in Table 1, where the A55 core consumes the least amount of power at a frequency of 500MHz, but is most energy efficient at 875MHz.

In the end, when the current load is below L_T, the optimal frequency would be the one with the lowest GCoE among those who could process the same volume without the projected load exceeding L_T.

As we defined the algorithm to determine the optimal frequency in both scenarios (performance and optimization), the result of our empirical functional and the quality of the current frequency estimation is determined as the distance between the current and optimal frequency indexes in a sorted frequency list.

3.2. Theoretical Foundation. In order for empirical functional to be usable in the SPSA2 algorithm and to prove the consistency of the algorithm, some conditions should be satisfied. First, the changes of optimal frequency are bounded by a task definition:

$$\|f_n - f_{n-1}\| \leqslant \delta < \infty.$$

[4]Besides, given that in our case U=1V, CoE and GCoE are numerically equal.

Then, $F(f)$ satisfies two assumptions:

Assumption 1: $F(f)$ is strongly convex and have a minimum point f^*:

$$\langle f - f^*, \nabla F(f) \rangle \geqslant \gamma \mathrm{ln}1.5 - 1, \forall \mathrm{f} \in \mathbb{R}.$$

Assumption 2:

The gradient $\nabla F(f)$ satisfies the Lipschitz condition:

$$\|\nabla F(f_1) - \nabla F(f_2)\| \leqslant \gamma 1.5^{\max(\mathrm{f}_1;\mathrm{f}_2)} \cdot \ln^2 1.5 \|\mathrm{f}_1 - \mathrm{f}_2\|, \\ \forall f_1, f_2 \in \mathbb{R}.$$

It is proved that SPSA2 converges when both assumptions for empirical functional are met [5], therefore, algorithm from 3.1 converges.

3.3. Strategies to Implement Observations. A straightforward approach to implementing SPSA2 takes 3 invocations of the DVFS routine. We note that due to the absence of floating point operations in the Android OS kernel, it is easier to manipulate with frequency indexes in a sorted array rather than with the frequencies themselves. In all cases if the index is beyond array boundaries, it is set to low or high boundary correspondingly.

1. Given the current frequency index i_i, a random number $\Delta = \pm 1$ is generated, and a frequency with the index $i_i + \Delta\beta$ is set to obtain the first noisy observation y^+ of functional.

2. Then the frequency with index $i_i - \Delta\beta$ is set to obtain the second noisy observation y^-.

3. New frequency is set by index $i_{i+1} = i_i - \frac{\alpha(y^+ - y^-)}{2\Delta\beta}$.

However, by definition of SPSA2 a faster approach is also possible which takes only two invocations:

1. The current functional value under the current frequency with index i_i is treated as the first noisy observation y^0. A random number $\Delta = \pm 1$ is generated, and a frequency with the index $i_i + \Delta\beta$ is set to obtain the second noisy observation y^+ of functional.

2. New frequency is set by index $i_{i+1} = i_i - \frac{\alpha(y^+ - y^0)}{\Delta\beta}$.

However, waiting for 2 or 3 invocations to be finished could be detrimental to governor performance and estimation accuracy. Therefore, we try out another scheme, which is not strict in terms of the SPSA2 definition, but takes only a single iteration:

1. Given the current frequency index i_i, a random number $\Delta = \pm 1$ is generated. The load for frequencies with indexes $i_i \pm \Delta\beta$ is calculated in the assumption that the volume will remain the same, and the functional value for

the modelled observations y^+ and y^- is obtained. Then the new frequency is set by index $i_{i+1} = i_i - \frac{\alpha(y^+ - y^-)}{2\Delta\beta}$.

In the last scheme, we also added an optimization for the performance mode. When the observed load is 100%, we modify the value of β and thus introduce a sequence of β_n instead of a single value to increase frequency more substantially. If, however, this decision would prove excessive, and the high load would not last long, the next iteration of the algorithm would reduce operating frequency.

In all schemes, additional precautions are taken before setting i_{i+1} to properly handle corner cases. Based on the number of observations, we will further call those schemes SPSA2$_3$, SPSA2$_2$ and SPSA2$_1$ respectively.

3.4. Accounting for Frequency Change Lag. The time CPU takes to change frequency could be considerable. For example, the `OnDemand` governor is scheduled to re-evaluate optimal frequency every 10 ms. Historically, `OnDemand` was developed for desktop computers and servers, and frequency switch could happen within 10 ms. However, in our preliminary research with Xiaomi Redmi Note 8 Pro, we found out that after the initial frequency change request by the `OnDemand`-based DVFS governor, it could take up to 3 consequent DVFS routine calls to register the updated frequency from the CPU, and the DVFS governor sends a frequency change request on every invocation.

This situation is indicative that even the most commonly available DVFS governors are based on assumptions that may not be true for the underlying hardware.

As our SPSA2 governors are also using the `OnDemand` governor timer model, and it is important for the algorithm to make noisy observations at the requested frequencies, we skip the DVFS routine invocations where the CPU cluster frequency is still different from the one requested previously. The algorithm continues when a proper frequency is set, and it is assumed that the load between two routine invocations was obtained at the requested frequency; thus, the noisy load observation is properly obtained.

3.5. EAS Integration. Energy Aware Scheduling works closely with the `schedutil` DVFS governor, and it is stated that EAS is not guaranteed to reduce energy consumption if the `schedutil` is not an active governor. In fact, `schedutil` implements the same strategy as the `OnDemand` governor, that is, it selects cluster frequency proportional to the maximum load over cluster cores with the minimum frequency corresponding to 0% load and the maximum frequency corresponding to 100% load. However, the EAS CPU load definition is significantly different from the one used in `OnDemand`. The CPU load is estimated in terms of the EAS energy model and occupancy of all tasks assigned to a specific CPU core. Instead of a posterior evaluation, it is estimated a priori.

For our proposed schemes we used the `OnDemand` load definition even when EAS is turned on, and, as shown in the experiments, tuning α, β and L_T is enough to align our SPSA2 governors with EAS.

4. Experimental Methodology

4.1. Device Selection. For our experiments, we used Samsung Galaxy s7 SM-G930F. Its processor, Exynos 8 Octa (8890) has 2 clusters with 4 Cortex-A53 cores (LITTLE) and 4 Exynos M1 cores (big). It is notable that while originally supporting Android 8, it could be patched to run Android 10 and 11.

The smartphone was patched to run the herolte[5] Android 11 kernel, which could be run on the test device with the EAS scheduler. To test the default CFS scheduler, we used Samsung android_kernel_samsung_universal8890 for Android 10[6]. This kernel selection allows us to compare our governors' performance against commonly available Interactive, OnDemand and Schedutil governors. Our DVFS governors were added as additional modules to both of those kernels, and it was required to obtain root access for the smartphone to install custom builds. Switching to our governors was done by the default `cpufreq` system calls [9].

4.2. Test Cases. As noted in our previous work [4], it is important for the general-purpose DVFS governor to be able to handle different kinds of workloads while providing a trade-off between performance and energy consumption. However, each governor is built with a particular workload model in mind, and this model is not bound to reflect real-world scenarios.

We've considered the following list of diverse test cases to be implemented. Each test case was assigned a name for further reference.

1. **videoVLC** – playing MP4 file using VLC video player, preliminarily selected as a default video player.
2. **trialXTreme3** – imitation of playing Trial Xtreme 3.
3. **flappyBird** – imitation of playing Flappy Bird.
4. **type** – creating a note in the Notes application, writing text and deleting a note.
5. **camera** – launching a default camera application and recording a video with it. The resulting video is deleted in the end.

[5]https://github.com/pascua28/herolt

[6]https://github.com/8890q/android_kernel_samsung_universal8890/tree/lineage-17.1

6. **twitch** – watching a Twitch stream in a browser and closing the browser after a while[7].

As the `Monkeyrunner` tool we used previously was not available in the toolchain for the test device, a set of utilities that mirrors its functionality to run tests was written in Python[8].

All tests are launched 10 times for 5 minutes to average the influence of system background processes. They are implemented as Python scripts as well. To launch a test, a smartphone needs to be connected to a controlling PC, and the test execution is controlled through `adb` commands.

To test smartphone performance we selected Geekbench 5.5.1, as it has an open methodology we found valid for our purposes [23]. The performance tests were launched 3 times for each evaluated governor due to their longevity, and an average of the obtained score points was calculated.

4.3. Modified Energy Consumption Models. Overall, direct measurement approaches were not used in the research for this paper due to the difficulties outlined above. While the basic energy consumption model could already be used, it does not take into account idle state management. Moreover, the source code behind `time-in-state` files is not synchronized with the idle governors. To address this issue, we propose the following modifications to the basic model.

Equation (2) implicitly assumes that the CPU consumes a fixed amount of power by just being at a specific frequency, regardless of the fact if there is an active computational process or not. This assumption is supported by the implementation of idle task while the CPU is in C0 (Active) state. When the OS scheduler cannot schedule an active tasks for the core, it assigns a special idle task, which consists of `NOP` (No Operation) instructions. C1, however, involves stopping CPU core timers, and the computational process cannot be run.

Given than, only C0 and C1 states are available on the test device, and given the fact that there could be several clusters of cores with a single frequency for all cores within a cluster, we modify Equation (2) as:

$$E = \sum_{i=1}^{Ncl} Eidle_i(t) + \sum_{i=1}^{Ncl} U_i \cdot \sum_{j=1}^{Nf_i} I_i(f_j) \cdot \sum_{k=1}^{Ncores_i} t_{fjk}, \tag{3}$$

[7]We would like to thank @StreamerHouse channel for their dedication to 24/7 streaming schedule as it allowed to keep test case source code without modifications regardless of the launch time.

[8]https://github.com/makar-pelogeiko/freq _gov_test

where Ncl is a number of core clusters, $Eidle_i(t)$ is energy consumed by a CPU for just being turned on, Nf_i is a number of frequencies for ith cluster, Ui is a nominal voltage of cores in ith cluster, $I_i(f)$ is a nominal current for slected frequency f in ith cluster, $Ncores_i$ is a number of cores in ith cluster, t_{fjk} is a time spent by kth core within ith cluster on jth frequency.

While this model is more elaborate than the one described in (2), it could be simplified for practical reasons. In the first term, we assume that each cluster has linear energy consumption over time for just being turned on. Intuitively, it seems that to properly calculate idle state energy footprint we should split $Eidle_i(t)$ to basic CPU energy consumption and C1 footprint for each core, but base CPU and C1 idle state power profiles are not available in `power_profile.xml`, we omit this term from further analysis.

Moreover, calculating $\sum_{k=1}^{Ncores_i} t_{fjk}$ seems excessive, because cores of the same cluster operate at the same frequency. However, this term becomes useful when we are trying to estimate the impact of deep sleep idle states impact on energy consumption.

While idle state stops CPU core timers, it does not change the nominal core operating frequency, that is, a core operating at 1 GHz before sleep will continue to work at 1 GHz after sleep. We initially assumed that the core is put to sleep when there are not enough tasks to schedule in the system overall, so the DVFS governor sets the minimum operating frequency for the cluster. Our initial idea was to deduct idle time in C1 from the time spent by the core on the lowest frequency such as in:

$$E = \sum_{i=1}^{Ncl} U_i \cdot (\sum_{k=1}^{Ncores_i} I_i(f_1)(t_{f0k} - t_k^{idle}) + \sum_{j=2}^{Nf_i} I_i(f_j) t_{fjk}),$$

where t_k^{idle} is the time spent by kth core in idle states. However, this assumption proved to be wrong, as it was shown experimentally that there were cases where $t_{f0k} < t_k^{idle}$), which means that the system put a core to sleep at a higher frequency than minimal.

While it is technically possible to modify the Android OS kernel and to track at which frequency the core was suspended, instead we assume that idle state management is independent of DVFS, but not of the OS scheduler, and a core may be put to sleep at any frequency at any time. Therefore, we propose to track idle state energy footprint by deducting idle state time from each time spent on frequency proportionally:

$$E = \sum_{i=1}^{Ncl} U_i \cdot \sum_{j=1}^{Nf_i} I_i(f_j) \cdot \sum_{k=1}^{Ncores_i} (t_{fjk} - t_k^{idle} \frac{t_{fjk}}{\sum_{m=1}^{Nf_i} t_{fmk}}).$$

As this approach is not precise but gives empirically good insights on actual energy expenditure, we use both original basic and modified models into report our results.

4.4. Device Preparation. Validity of experiments and output stability was enhanced by the following protocol we have described in the previous research [7]:

– All unnecessary applications were uninstalled, and all application activities were turned off for those that could not be uninstalled.

– Peripherals that were not used in a particular test case (i.e. Wi-Fi, 4G, GPS) were turned off.

– A cool-down period of 2 minutes was taken between the tests for background processes to finalize and decrease the device temperature.

– Before running a particular test case, it was run for a single time for warming-up purposes.

Battery charge levels were not homogenized before each test runs, as it was unnecessary by our energy consumption models. In fact, for the device to be controlled by a test run utility, it was connected to a PC via USB, so the battery was charging during the test runs.

5. Experiments. The governor implementations are available for Android 11[9] and Android 10[10].

The full experimental data is also available.[11]

The first set of the experiments was taken under Android 11 with EAS turned on. By enumerating possible parameters we found out the best-performing settings for all SPSA algorithms:

1. $\alpha = 2, \beta = 1, L_T = 70\%$;

2. Cluster 0: $\alpha = 2$, $\beta = 1$, $L_T = 80\%$; Cluster 1: $\alpha = 3$, $\beta = 1$, $L_T = 98\%$.

Tables 2 to 5 contain the median values for energy consumption under the basic and modified energy models. Here and in the subsequent tables we

[9]https://github.com/makar-pelogeiko/herolte _Eas _Idle _modification/tree/Q-stable-spsa_gov

[10]https://github.com/makar-pelogeiko/android _kernel _samsung _universal8890/tree/lineage-17.1-spsa _gov

[11]https://studentspburu-my.sharepoint.com/personal/st076963 _student _spbu _ru/ _layouts/15/ onedrive.aspx?id=%2Fpersonal%2Fst076963%5Fstudent%5Fspbu%5Fru%2FDocuments% 2Fdiploma%2Fresultsga=1

highlight scenarios where the SPSA governor energy consumption is higher than energy consumption of every baseline governor in *italic* and the cases where SPSA is better than some of the baseline governors in **bold**. Table 6 contains performance data. The first line for SPSA governors corresponds to the first set of parameters, and the second line – to the second set.

Table 2. Energy consumption (mAh) under Android 11 with EAS scheduler (basic energy model), videoVLC, trialXTreme3 and flappyBird tests

Algorithm	videoVLC	trialXTreme3	flappyBird
$SPSA2_3$	21.67	27.62	24.86
	21.43	24.53	24.62
$SPSA2_2$	23.40	37.44	40.64
	22.65	32.50	34.36
$SPSA2_1$	23.75	43.43	**47.04**
	22.17	30.43	31.23
Schedutil	24.56	44.30	43.56
Interactive	141.21	119.08	104.73
OnDemand	34.86	86.06	92.86

Table 3. Energy consumption (mAh) under Android 11 with EAS scheduler (basic energy model), type, camera and twitch, tests

Algorithm	type	camera	twitch
$SPSA2_3$	32.61	22.43	30.59
	30.91	22.02	24.86
$SPSA2_2$	**46.46**	25.27	38.62
	39.96	23.67	28.93
$SPSA2_1$	**52.19**	26.44	47.06
	35.66	22.86	27.32
Schedutil	47.09	27.36	51.83
Interactive	122.31	55.36	91.33
OnDemand	97.43	56.16	80.68

Overall, the energy consumption dispersion is omitted in this paper, but it is available in the full experimental report. It should be noted that the values for the SPSA and `schedutil` governors have a visibly low dispersion in all experiments, and energy consumption could be compared by the median value only. It is not true for `OnDemand` and especially `Interactive` governors. For example, `Interactive` showed in the videoVLC test the minimum and maximum values under the basic energy model of 42.42 and 155.91 mAh respectively. However, it does not affect the validity of the analysis, as even the minimum energy consumption of both governors was higher than the maximum value for `schedutil` or SPSA governors.

Table 4. Energy consumption (mAh) under Android 11 with EAS scheduler (modified energy model), videoVLC, trialXTreme3 and flappyBird tests

Algorithm	videoVLC	trialXTreme3	flappyBird
$SPSA2_3$	5.90	14.71	10.23
	6.11	13.49	10.15
$SPSA2_2$	**6.34**	19.16	**16.27**
	6.24	16.71	14.16
$SPSA2_1$	**6.31**	19.68	**18.21**
	5.93	15.61	12.75
Schedutil	5.91	20.24	15.70
Interactive	33.74	46.78	35.46
OnDemand	8.59	34.84	32.48

Table 5. Energy consumption (mAh) under Android 11 with EAS scheduler (modified energy model), type, camera and twitch tests

Algorithm	type	camera	twitch
$SPSA2_3$	8.99	14.41	22.84
	8.67	13.71	18.98
$SPSA2_2$	7.69	**17.18**	28.18
	12.30	15.24	21.88
$SPSA2_1$	**13.02**	**17.76**	33.25
	9.36	15.43	20.70
Schedutil	10.24	15.62	34.92
Interactive	21.83	30.56	58.48
OnDemand	21.35	25.07	47.53

Performance-wise, both existing and novel algorithms perform comparably. There are no clear outliers, and for single-core test, the difference between the best and worst performance is within 15%, while it is within 18% for the multicore scenario. We note that the first set of SPSA parameters provides better performance than the second set. $SPSA2_3$ performance is worsened by the fact it needs 3 observations, but not too much.

Table 6. GeekBench 5.5.1 performance scores

Algorithm	Single core	Multicore
SPSA2_3	287	970
	282	913
SPSA2_2	302	1025
	290	962
SPSA2_1	331	1073
	324	966
Schedutil	319	970
Interactive	321	1112
OnDemand	296	1068

When we look at energy consumption data, it should be noted that the modified energy model significantly changes the observed values compared to the basic model, and in some cases their ratio. The modified model, in our opinion, still better reflects data reality if direct measurement is not available, and our next analysis is based on it, even though the basic model is more complimentary to the SPSA governors.

The `Interactive` and `OnDemand` governors are consistently more power-hungry than `schedutil` under EAS, and, as a general rule, we definitely suggest using `schedutil` over other governors if EAS is available in stock firmware.

As for the SPSA governors, all of them could be used in place of `schedutil`. First of all, they all handle trialXTreme3 and twitch tests better than schedutil. We assume it to be related to $\beta = 1$ in our experiments, as higher β results in a more rapid gradient descent, and in a quickly changing reality of CPU load complicated by frequency switch lags relatively smooth frequency change is good for consistent CPU usage. SPSA2_1 with the first set of parameters is shown to consume more energy in most of the tests (up to 27$\%$ for type test), while being only 10$\%$ better in performance compared to `schedutil`, so the second set of parameters is recommended there, as performance is at the same level, while energy consumption may be up to 31$\%$ better. SPSA2_2 is the most balanced governor, as it is up to 20$\%$ better in some tests and up to 10$\%$ worse in others energy-wise, but it demonstrates a 5.6$\%$ performance increase in multicore case, which is more realistic scenario. SPSA2_3 is the most conservative of other governors energy and performance-wise, but overall multicore performance is comparable to `schedutil`.

A special case is the videoVLC test, where the SPSA governors tend to spend more energy than `schedutil`. The difference is never significant (no more than 7.2% for SPSA2_2), but our explanation is that the timings of intense computations (frame decoding) are such that the SPSA governors cannot drop frequency to lower values. The only exception is SPSA2_3, but the difference is within a margin of a statistical error.

Tables 7 to 10 contain the median values for energy consumption under the basic and modified energy models for the tests run under Android 10 and CFS scheduler.The situation there is drastically different. First, `schedutil` cannot be used there as a benchmark as it would be running under the conditions it was not designed for. Then, the `Interactive` governor demonstrates a remarkable improvement in energy consumption.

Table 7. Energy consumption (mAh) under Android 10 with CFS scheduler (basic energy model), videoVLC, trialXTreme3 and flappyBird tests

Algorithm	videoVLC	trialXTreme3	flappyBird
SPSA2_3	**23.52**	**49.49**	48.67
	23.55	43.87	44.68
SPSA2_2	**26.14**	**66.62**	**71.74**
	24.34	**63.82**	**62.85**
SPSA2_1	**26.58**	*101.24*	*116.08*
	23.65	**57.40**	**85.69**
Interactive	22.25	44.31	53.47
OnDemand	29.18	76.41	94.63

Table 8. Energy consumption (mAh) under Android 10 with CFS scheduler (basic energy model), type, camera and twitch tests

Algorithm	type	camera	twitch
SPSA2_3	57.73	**26.25**	38.71
	51.96	**25.08**	30.22
SPSA2_2	**81.87**	**36.10**	**45.66**
	70.70	**29.28**	33.41
SPSA2_1	*116.36*	**31.28**	**50.54**
	68.21	**25.23**	32.04
Interactive	59.52	24.32	41.73
OnDemand	101.57	41.39	66.24

However, all SPSA governors demonstrate energy behavior reminiscent of SPSA1 governor [4], where they are most of the time either between `OnDemand` and `Interactive`, and in some margin cases inconsistently demonstrate better or worse results.

Table 9. Energy consumption (mAh) under Android 10 with CFS scheduler (modified energy model), videoVLC, trialXTreme3 and flappyBird tests

Algorithm	videoVLC	trialXTreme3	flappyBird
$SPSA2_3$	**5.70**	**13.51**	12.07
	5.57	11.13	11.07
$SPSA2_2$	**6.35**	**17.53**	**17.27**
	5.66	**14.57**	**15.38**
$SPSA2_1$	**6.20**	**19.83**	*25.58*
	5.44	11.67	**20.35**
Interactive	1.31	13.28	12.30
OnDemand	7.08	21.65	24.03

Table 10. Energy consumption (mAh) under Android 10 with CFS scheduler (modified energy model), type, camera and twitch tests

Algorithm	type	camera	twitch
$SPSA2_3$	**13.22**	**4.88**	16.53
	11.33	**4.48**	13.54
$SPSA2_2$	**17.44**	**4.88**	**18.98**
	15.22	**4.18**	14.61
$SPSA2_1$	*23.59*	**3.27**	**20.50**
	14.63	2.14	13.91
Interactive	11.71	2.93	16.92
OnDemand	22.03	4.98	23.61

The reason behind such energy behavior is a change in scheduling strategy. Unlike EAS, CFS does not prioritize LITTLE cores over big ones, and in similar situations, big cores are loaded more resulting in a higher power drain.

Additionally, while overall $SPSA2_3$ demonstrates an acceptable energy consumption, its performance is worse, as at load levels of 99–100% it does not immediately recommend increase frequency. As other algorithms demonstrate higher overall energy consumption than `Interactive`, and the latter provides an acceptable performance to the end user, we did not run performance tests for the SPSA governors.

6. Conclusion. Our experiments show that all SPSA governors could be used alongside EAS to obtain better CPU energy efficiency in common use cases. In CFS-controlled environments, the SPSA governors could also be used, but they demonstrate energy efficiency comparable to existing governors.

We conclude, that like SPSA1 in our previous work [4, 7], the SPSA2 family of governors performs better than `OnDemand`, `Interactive` and `schedutil` in the scenarios where the workload is evenly distributed over time or has prolonged periods of calculations – improvement of up to 31% could be observed with the same performance under EAS. We observe that stochastic optimization provides good optimal point tracking in noisy environments in a long run without setting seemingly optimal frequencies at every step. In the spiky workloads, a conservative approach to frequency changes causes those computational spikes to be treated as constant load and therefore could result in a somewhat higher CPU energy consumption. In the end, the SPSA2 family could be used to prolong a smartphone's lifetime on a day-to-day basis.

References

1. Number of smartphone mobile network subscriptions worldwide from 2016 to 2022, with forecasts from 2023 to 2028. Available at: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. (accessed 10.05.2023).
2. Spall J.C. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control. 1992. vol. 37. no. 3. pp. 332–341.
3. Granichin O., Amelina N. Simultaneous perturbation stochastic approximation for tracking under unknown but bounded disturbances. IEEE Transactions on Automatic Control. 2015. vol. 60. no. 6. pp. 1653–1658.
4. Bogdanov E., Bozhnyuk A., Bykov D., Sartasov S., Sergeenko A., Granichin O. Dynamic Voltage-Frequency Optimization using Simultaneous Perturbation Stochastic Approximation. 60th IEEE Conference on Decision and Control (CDC). 2021. pp. 3774–3779.
5. Granichin O., Vakhitov A. Accuracy for the SPSA algorithm with two measurements. WSEAS Transactions on Systems. 2006. vol. 5.
6. Mair H.T., Gammie G., Wang A., Lagerquist R., Chung C.J., Gururajarao S., Kao P., Rajagopalan A., Saha A., Jain A., Wang E., Ouyang S., Wen H., Thippana A., Chen HsinChen, R.S., Chau M., Varma A., Flachs B., Peng M., Tsai A., Lin V., Fu U., Kuo W., Yong L.-K., Peng C., Shieh L., Wu J., Ko U. 4.3 A 20nm 2.5GHz ultra-low-power tri-cluster CPU subsystem with adaptive power allocation for optimal mobile SoC performance. 2016 IEEE International Solid-State Circuits Conference (ISSCC). 2016. pp. 76–77.
7. Bogdanov E., Bozhnyuk A., Sartasov S., Granichin O. On Application of Simultaneous Perturbation Stochastic Approximation for Dynamic Voltage-Frequency Scaling in Android OS. 7th International Conference on Event-Based Control, Communication and Signal Processing (EBCCSP'21). 2021. DOI: 10.1109/EBCCSP53293.2021.9502396.
8. The kernel development community. Energy Aware Scheduling. Available at: https://www.kernel.org/doc/html/next/scheduler/sched-energy.html. (accessed 10.05.2023).
9. CPU frequency and voltage scaling code in the Linux (TM) kernel. Linux CPUFreq. CPUFreq Governors. Available at: https://android.googlesource.com/kernel/common/+/a7827a2a60218b25f222b54f77ed38f57aebe08b/Docum freq/governors.txt. (accessed 10.05.2023).

10. Basireddy K.R., Singh A.K., Al-Hashimi B.M., Merrett G.V. AdaMD: Adaptive Mapping and DVFS for Energy-Efficient Heterogeneous Multicores. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2020. vol. 39. no. 10. pp. 2206–2217.
11. Lee J., Nam S., Park S. Energy-Efficient Control of Mobile Processors Based on Long Short-Term Memory. IEEE Access. 2019. vol. 7. pp. 80552–80560.
12. Rapp M., Krohmer N., Khdr H., Henkel J. NPU-accelerated imitation learning for thermal- and QoS-aware optimization of heterogeneous multi-cores. Proceedings of the 2022 Conference and Exhibition on Design, Automation and Test in Europe (DATE '22). 2021. pp. 584–587.
13. Kim S., Bin K., Ha S., Lee K., Chong S. ZTT: learning-based DVFS with zero thermal throttling for mobile devices. Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'21). 2021. pp. 41–53.
14. Song S., Kim J., Chung J.-M. Energy Consumption Minimization Control for Augmented Reality Applications based on Multi-core Smart Devices. IEEE International Conference on Consumer Electronics (ICCE). 2019. DOI: 10.1109/ICCE.2019.8661917.
15. Ohk S.-R., Kim Y., Kim Y.-J. Phase-Based Low Power Management Combining CPU and GPU for Android Smartphones. Electronics. 2022. vol. 11. no. 16. DOI: 10.3390/electronics11162480.
16. Dey S., Isuwa S., Saha S., Singh A.K., McDonald-Maier K. CPU-GPU-Memory DVFS for Power-Efficient MPSoC in Mobile Cyber Physical Systems. Future Internet. 2022. vol. 14. no. 3. DOI: 10.3390/fi14030091.
17. CPU Idle Time Management. Available at: https://docs.kernel.org/admin-guide/pm/cpuidle.html. (accessed 10.05.2023).
18. Metri G., Agrawal A., Peri, R., Brockmeyer M., Weisong S. A simplistic way for power profiling of mobile devices. 2012 International Conference on Energy Aware Computing. 2012. DOI: 10.1109/ICEAC.2012.6471020.
19. Monsoon Power Monitor Specifications. Available at: https://www.msoon.com/specifications. (accessed 10.05.2023).
20. Chung Y., Lin C., King C. ANEPROF: Energy Profiling for Android Java Virtual Machine and Applications. 2011 IEEE 17th International Conference on Parallel and Distributed Systems. 2011. pp. 372–379. DOI: 10.1109/ICPADS.2011.28.
21. Measuring Component Power. Available at: https://source.android.com/docs/core/power/component. (accessed 10.05.2023).
22. Granichin O. Linear regression and filtering under nonstandard assumptions (arbitrary noise). IEEE Transactions on Automatic Control. 2004. vol. 49. no. 10. pp. 1830–1837.
23. Geekbench 5 CPU Workloads. Available at: https://www.geekbench.com/doc/geekbench5-cpu-workloads.pdf. (accessed 10.05.2023).

Pelogeiko Makar – Student, Software engineering department, faculty of mathematics and mechanics, St. Petersburg State University (SPbSU). Research interests: energy-efficient programming. m.pelogeiko@mail.ru; 28, Universitetsky Av., 198504, St. Petersburg, Russia; office phone: +7(812)428-4910.

Sartasov Stanislav – Assistant professor of software engineering department, Faculty of mathematics and mechanics, St. Petersburg State University (SPbSU); chief technology officer, Denominator.One. Research interests: sustainable software development, software energy efficiency, industrial software engineering, biometrics. The number of publications — 10. stanislav.sartasov@yandex.ru; 28, Universitetsky Av., 198504, St. Petersburg, Russia; office phone: +7(812)428-4910.

Granichin Oleg – Ph.D., Dr.Sci., Professor of software engineering department, Faculty of mathematics and mechanics, St. Petersburg State University (SPbSU); Laboratory "Control of complex systems", Institute for Problems in Mechanical Engineering. Research interests: randomized optimization and estimation algorithms, stochastic optimization in computer science, adaptive and optimal control, pattern recognition. The number of publications — 100. oleg_granichin@mail.ru; 28, Universitetsky Av., 198504, St. Petersburg, Russia; office phone: +7(812)428-4910.

Acknowledgements. This work was supported in part by the St. Petersburg State University (project ID 94062114).

УДК 004.451.25 **DOI** 10.15622/ia.22.5.3

М.А. Пелогейко, С.Ю. Сартасов, О.Н. Граничин

О СТОХАСТИЧЕСКОЙ ОПТИМИЗАЦИИ ЭНЕРГОПОТРЕБЛЕНИЯ ПРОЦЕССОРА СМАРТФОНА

Пелогейко М.А., Сартасов С.Ю., Граничин О.Н **О стохастической оптимизации энергопотребления процессора смартфона.**

Аннотация. Увеличение времени работы смартфона – это постоянное стремление, которое с каждым годом становится все более и более важным. Это может быть достигнуто с помощью более совершенного оборудования или путем внедрения в программное обеспечение практик с учетом энергопотребления, и последний подход является более доступным. Поскольку ЦП является одним из самых энергоемких устройств для смартфонов, динамическое масштабирование частоты напряжения (DVFS) представляет собой метод настройки частоты ЦП в соответствии с текущими вычислительными потребностями, и уже были разработаны различные алгоритмы, как энергосберегающие, так и энергонезависимые. Следуя нашей предыдущей работе по этому вопросу, мы предлагаем новый подход DVFS для использования стохастической аппроксимации одновременных возмущений (SPSA) с двумя зашумленными наблюдениями для отслеживания оптимальной частоты и реализации нескольких алгоритмов на его основе. Кроме того, мы также решаем проблему аппаратной задержки между сигналом для ЦП об изменении частоты и ее фактическим обновлением. Поскольку ОС Android может использовать планировщик задач по умолчанию или планировщик с учетом энергопотребления, который способен использовать преимущества разнородных архитектур мобильных ЦП, таких как ARM big.LITTLE, мы также исследуем схему интеграции между предлагаемыми алгоритмами и планировщиками ОС. Представлена методология тестирования на основе моделей для сравнения разработанных алгоритмов с существующими, а также описан набор тестов, отражающий реальные сценарии использования. Наши эксперименты показывают, что алгоритм на основе SPSA хорошо работает с EAS с упрощенной схемой интеграции, демонстрируя производительность ЦП, сравнимую с другими алгоритмами DVFS с учетом энергопотребления, и снижение энергопотребления.

Ключевые слова: ОС Android, динамическое масштабирование частоты напряжения, стохастическая оптимизация, SPSA, энергопотребление.

Литература

1. Number of smartphone mobile network subscriptions worldwide from 2016 to 2022, with forecasts from 2023 to 2028. Available at: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. (accessed 10.05.2023)
2. Spall J.C. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control. 1992. vol. 37. no. 3. pp. 332–341.
3. Granichin O., Amelina N. Simultaneous perturbation stochastic approximation for tracking under unknown but bounded disturbances. IEEE Transactions on Automatic Control. 2015. vol. 60. no. 6. pp. 1653–1658.
4. Bogdanov E., Bozhnyuk A., Bykov D., Sartasov S., Sergeenko A., Granichin O. Dynamic Voltage-Frequency Optimization using Simultaneous Perturbation Stochastic

Approximation. 60th IEEE Conference on Decision and Control (CDC). 2021. pp. 3774–3779.

5. Granichin O., Vakhitov A. Accuracy for the SPSA algorithm with two measurements. WSEAS Transactions on Systems. 2006. vol. 5.

6. Mair H.T., Gammie G., Wang A., Lagerquist R., Chung C.J., Gururajarao S., Kao P., Rajagopalan A., Saha A., Jain A., Wang E., Ouyang S., Wen H., Thippana A., Chen HsinChen, R.S., Chau M., Varma A., Flachs B., Peng M., Tsai A., Lin V., Fu U., Kuo W., Yong L.-K., Peng C., Shieh L., Wu J., Ko U. 4.3 A 20nm 2.5GHz ultra-low-power tri-cluster CPU subsystem with adaptive power allocation for optimal mobile SoC performance. 2016 IEEE International Solid-State Circuits Conference (ISSCC). 2016. pp. 76–77.

7. Bogdanov E., Bozhnyuk A., Sartasov S., Granichin O. On Application of Simultaneous Perturbation Stochastic Approximation for Dynamic Voltage-Frequency Scaling in Android OS. 7th International Conference on Event-Based Control, Communication and Signal Processing (EBCCSP’21). 2021. DOI: 10.1109/EBCCSP53293.2021.9502396.

8. The kernel development community. Energy Aware Scheduling. Available at: https://www.kernel.org/doc/html/next/scheduler/sched-energy.html. (accessed 10.05.2023).

9. CPU frequency and voltage scaling code in the Linux (TM) kernel. Linux CPUFreq. CPUFreq Governors. Available at: https://android.googlesource.com/kernel/common/+/a7827a2a60218b25f222b54f77ed38f57aebe08b/Docum freq/governors.txt. (accessed 10.05.2023).

10. Basireddy K.R., Singh A.K., Al-Hashimi B.M., Merrett G.V. AdaMD: Adaptive Mapping and DVFS for Energy-Efficient Heterogeneous Multicores. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2020. vol. 39. no. 10. pp. 2206–2217.

11. Lee J., Nam S., Park S. Energy-Efficient Control of Mobile Processors Based on Long Short-Term Memory. IEEE Access. 2019. vol. 7. pp. 80552–80560.

12. Rapp M., Krohmer N., Khdr H., Henkel J. NPU-accelerated imitation learning for thermal- and QoS-aware optimization of heterogeneous multi-cores. Proceedings of the 2022 Conference and Exhibition on Design, Automation and Test in Europe (DATE ’22). 2021. pp. 584–587.

13. Kim S., Bin K., Ha S., Lee K., Chong S. ZTT: learning-based DVFS with zero thermal throttling for mobile devices. Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys’21). 2021. pp. 41–53.

14. Song S., Kim J., Chung J.-M. Energy Consumption Minimization Control for Augmented Reality Applications based on Multi-core Smart Devices. IEEE International Conference on Consumer Electronics (ICCE). 2019. DOI: 10.1109/ICCE.2019.8661917.

15. Ohk S.-R., Kim Y., Kim Y.-J. Phase-Based Low Power Management Combining CPU and GPU for Android Smartphones. Electronics. 2022. vol. 11. no. 16. DOI: 10.3390/electronics11162480.

16. Dey S., Isuwa S., Saha S., Singh A.K., McDonald-Maier K. CPU-GPU-Memory DVFS for Power-Efficient MPSoC in Mobile Cyber Physical Systems. Future Internet. 2022. vol. 14. no. 3. DOI: 10.3390/fi14030091.

17. CPU Idle Time Management. Available at: https://docs.kernel.org/admin-guide/pm/cpuidle.html. (accessed 10.05.2023).

18. Metri G., Agrawal A., Peri, R., Brockmeyer M., Weisong S. A simplistic way for power profiling of mobile devices. 2012 International Conference on Energy Aware Computing. 2012. DOI: 10.1109/ICEAC.2012.6471020.

19. Monsoon Power Monitor Specifications. Available at: https://www.msoon.com/specifications. (accessed 10.05.2023).
20. Chung Y., Lin C., King C. ANEPROF: Energy Profiling for Android Java Virtual Machine and Applications. 2011 IEEE 17th International Conference on Parallel and Distributed Systems. 2011. pp. 372–379. DOI: 10.1109/ICPADS.2011.28.
21. Measuring Component Power. Available at: https://source.android.com/docs/core/power/component. (accessed 10.05.2023).
22. Granichin O. Linear regression and filtering under nonstandard assumptions (arbitrary noise). IEEE Transactions on Automatic Control. 2004. vol. 49. no. 10. pp. 1830–1837.
23. Geekbench 5 CPU Workloads. Available at: https://www.geekbench.com/doc/geekbench5-cpu-workloads.pdf. (accessed 10.05.2023).

Пелогейко Макар Андреевич — студент, кафедра разработки программного обеспечения, факультет математики и механики, Санкт-Петербургский государственный университет (СПбГУ). Область научных интересов: энергоэффективное программирование. m.pelogeiko@mail.ru; Университетский проспект, 28, 198504, Санкт-Петербург, Россия; р.т.: +7(812)428-4910.

Сартасов Станислав Юрьевич — доцент кафедры разработки программного обеспечения, факультет математики и механики, Санкт-Петербургский государственный университет (СПбГУ); главный технический директор, Denominator.One. Область научных интересов: устойчивая разработка программного обеспечения, энергоэффективность программного обеспечения, промышленная разработка программного обеспечения, биометрия. Число научных публикаций — 10. stanislav.sartasov@yandex.ru; Университетский проспект, 28, 198504, Санкт-Петербург, Россия; р.т.: +7(812)428-4910.

Граничин Олег Николаевич — д-р физ.-мат. наук, профессор кафедры разработки программного обеспечения, факультет математики и механики, Санкт-Петербургский государственный университет (СПбГУ); лаборатория “управление сложными системами”, Институт проблем машиноведения РАН. Область научных интересов: рандомизированные алгоритмы оптимизации и оценивания, стохастическая оптимизация в информатике, адаптивное и оптимальное управление, распознавание образов. Число научных публикаций — 100. oleg_granichin@mail.ru; Университетский проспект, 28, 198504, Санкт-Петербург, Россия; р.т.: +7(812)428-4910.

Поддержка исследований. — Работа была поддержана Санкт-Петербургским государственным университетом (проект № 94062114).