SAINT-PETERSBURG UNIVERSITY

Manuscript copyright

Sartasov Stanislav Yurievich

CENTRAL PROCESSOR ENERGY CONSUMPTION CONTROL BASED ON STOCHASTIC OPTIMIZATION

Scientific specialty 2.3.5. Mathematical methods and software for computer systems, complexes and computer networks

Thesis for the degree of Candidate of Technical Sciences

Translation from Russian

Scientific supervisor:
Doctor of Physical and Mathematical Sciences, Professor
Oleg Nikolaevich Granichin

St. Petersburg
2023

Contents

Introduction

At the moment, the task of optimizing the energy consumption of smartphones, tablets and other portable computers remains relevant both for their users and, in a broad sense, for the national economy. This is due to the fact that over the past 20 years the patterns of user access to information systems and network services have changed: to obtain necessary information, smartphones, tablets and other types of computers (hereinafter referred to as *mobile devices* or simply *devices*) that support autonomous operation are much more often used, and therefore battery life becomes one of the main criteria for the quality and convenience of a human-machine interface.

On the one hand, this problem is solved by the electronics and electrochemical industry, which develops both more capacious batteries and more energy-efficient components for such devices. The latter include central processing units (CPUs) based on a heterogeneous architecture. For example, according to the ARM big.LITTLE architecture, the CPU consists of two clusters of cores — less productive, but energy efficient (LITTLE cluster), and more productive and energy intensive (big cluster) [45]. However, it is also important whether the system and applied software is developed taking into account the peculiarities of the operation of these components from an autonomous power source.

One of the most researched power management technologies in operating systems (OS) is Dynamic Voltage Frequency Scaling (DVFS). The CPU is capable of operating at a finite set of operating frequencies and switching between them at run-time. A DVFS governor is an OS module that instructs the CPU to set one or another operating frequency depending on the observed state of the system as a whole and the frequency control algorithm. This module can work in conjunction with the OS task scheduler or be independent of it [103].

The task of optimizing CPU energy consumption is to maximize performance and minimize power drain, with the control signal being the setting of the CPU operating frequency. To solve this problem, various strategies are proposed, but at the same time, determining the best strategy is methodologically challenging.

Currently, there are two conceptually different approaches to measuring CPU energy consumption. The first one, the direct approach, consists of connecting a measuring device to a target component and analyzing the measurement results under various workloads. In addition to financial costs and the high requirements for qualification of researchers, this approach also has the disadvantage that its application is not always realistic, as to connect an external multimeter to the power circuits of a real device without disturbing its performance might be impossible. The idea of indirect approach is to measure some characteristics indirectly related to power consumption (the time the CPU is at a certain frequency, the number of instructions of one kind or another in the workload), and convert them into power consumption data using a mathematical model. This approach is much cheaper and less demanding, but the accuracy of the estimate is directly related to the quality of the model [73]. The models that exist today do not fully describe the behavior of modern heterogeneous CPUs. Thus, the task of creating new models and methods for measuring the power consumption of the CPU of a mobile device is relevant to assess the quality of the software modules that optimize power consumption.

To date, several approaches have been developed to optimize the power consumption of the CPU of a mobile device. X. Li, W. Wen and X. Wang suggested to include special energy profile of a particular functionality in the DVFS subsystem before using it [65]. The use of neural networks to predict the optimal frequency in situations similar to the current one was successfully used by Y.L. Chen, M.F. Chang, Ch.W. Yu, X.Zh. Chen, W.Y. Liang [19], J. Lee, S. Nam, and S. Park [60]. Transformation of the current state of the system, understood in a broad sense, into a predicted optimal frequency using an empirically established formula, is researched in the works of K. Poornambigai, M.L. Raj, P. Meena [81], L. Broyd, K. Nixon, X. Chen, H. Li, and Y. Chen [14]. Among the standard modules included in the distribution of the Android OS for mobile devices, the task of optimizing CPU power consumption is solved to a certain extent only in the schedutil governor, which is a part of the EAS system developed by Linaro and ARM [54]. The frequency is set in proportion to the estimate of energy load of heterogeneous CPU cluster based on the estimated computational intensity of the tasks assigned to each core of the cluster. This approach is more energy efficient than the rule-of-thumb idea behind OnDemand and Interactive governors, which do not take into account CPU power consumption. However, a common drawback of the proposed methods is insufficient elaboration of their theoretical foundations, especially in application to heterogeneous CPU architecture. An open question remains whether it is possible to further reduce power consumption while maintaining the performance and responsiveness of human-machine interfaces.

At the same time, an approach based on randomized recurrent stochastic optimization algorithms has been developed since the middle of twentieth century, which was initiated by the Robbins-Monroe and Kiefer-Wolfowitz procedures [56, 84]. This approach works well in many problems where measurements of target functional are noisy as shown, for example, N.O. Amelina and A.L. Fradkov in forming consensus on stochastic networks [5], by K.S. Amelin in controlling a group of quadrocopters [4], by Yu.V. Ivansky in solving the problem of differentiated consensus [6], by A.V. Gasnikov in determining equilibria in transport networks [33]. A detailed description of the approach is made in the monograph by O.N. Granichin and B.T. Poljak "Randomized estimation and optimization algorithms for almost arbitrary noise" [39]. In particular, one of the most general algorithms described in it is called Simultaneous Perturbation Stochastic Approximation (SPSA) and was researched by J. Spall [94], H.-F. Chen, T. Duncan, B. Pasik-Duncan [17], O.N. Granichin [36, 37], B.T. Polyak and A.B. Tsybakov [79]. The idea behind all variations of this algorithm is to approximate a gradient along a randomly chosen direction, and therefore SPSA is a type of stochastic gradient descent. The advantages of this approach include significantly weaker restrictions on measurement noise and the target functional, compared to traditionally used methods. In particular, in [36, 37] the efficiency of the algorithm is proved under almost arbitrary noise (for example, if the noise is "unknown-but-bounded").

The energy consumption optimization problem can be formalized in such a way as to satisfy the constraints of these algorithms. An attempt at such a formalization and an analysis of the obtained solution were published in the article by O.N. Granichin and V.E. Krasnoshchyokov on the applicability of stochastic optimization algorithms to optimizing the energy consumption of an MP3 player CPU [41]. The single-measurement SPSA algorithm can be applied to solve the problem of tracking the optimal operating frequency of a single-core CPU under a load of 0 to 3 processes. However, the result obtained does not scale to modern mobile devices with heterogeneous CPUs, where the number of simultaneously running processes does not have an upper limit. Until recently, there were no works devoted to the adaptation of stochastic optimization algorithms for this case.

The listed trends and problems confirm the relevance of the topic of the dissertation research.

The goal of the work is to develop mathematical methods and software for modern mobile operating systems to solve the problem of managing the energy consumption of a heterogeneous CPU when working in a wide range of computing loads. To achieve this goal, the following tasks were formulated and addressed:

1. Explore measurement methods and models for estimating the power consumption of the CPU of a mobile device, including heterogeneous architecture-based ones, and related features of program interaction.

2. Investigate the applicability of randomized algorithms for control, optimization and estimation of parameters for mathematical methods and OS software for managing energy consumption of a heterogeneous CPU and develop control algorithms based on them.

3. Create a DVFS module based on the developed control algorithms, integrate it into a modern mobile OS and evaluate the quality of the mobile device under its control, including human-machine interfaces, in conditions of parallel data processing.

Research methods. The dissertation uses methods of control theory, theory of optimization, theory of estimation, probabilities theory and mathematical statistics; methods of system programming, stochastic optimization, randomized algorithms are applied.

Main results:

1. A model for estimating the power consumption of a CPU built on a heterogeneous architecture is proposed and justified, taking into account the dynamic computational load and idle states management, and practical recommendations for its application are given.

2. An approach has been developed for solving the problem of power consumption management for a heterogeneous CPU based on randomized stochastic optimization algorithms, within which medium risk functionals for SPSA algorithms with one and two measurements are proposed and justified. Theoretical validity of the estimates provided by the developed algorithms is investigated and established within the limits imposed by the features of the CPU operation.

3. DVFS modules have been developed that optimize CPU power consumption, taking into account the peculiarities of its operation, based on the proposed functionals and SPSA algorithms with one and two measurements. A comparison with existing analogues has been made on the basis of prepared test benchmarks running in the Android OS. For modules based on SPSA algorithms with two observations, a certificate of state registration of computer programs has been obtained.

Scientific novelty. All main scientific results of the dissertation are new.

Theoretical value and practical significance. The theoretical value of the work is a study on the applicability of randomized stochastic optimization algorithms for solving the problem of tracking the optimal operating frequency of a heterogeneous CPU based on its power consumption model under arbitrary computing loads and identifying various noisy measurements usage strategies for them.

The developed software, as well as the principles behind it, can be applied to solve similar problems of CPU frequency control based on other optimization criteria. The introduced conceptual apparatus can be applied in the development of energy-efficient software at both the applied and system levels, as well as for supporting various software.

The system software created during the dissertation research increases the energy efficiency of data and knowledge processing in computers with heterogeneous CPUs and can be ported to other OS builds, as well as serve as the basis for other DVFS modules.

Research validation. The results of the dissertation were presented at the Fourth Conference on Software Engineering and Information Management (SEIM'20) (St. St. Petersburg, Russia, May 16, 2020), at the Fifth Conference on Software Engineering and Information Management (SEIM'21) (St. Petersburg, Russia, April 17, 2021), at the IEEE 7th International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP'21) (June 23–25, 2021, Krakow, Poland), at the 60th IEEE conference on Decision and Control (CDC'21) (December 13–17, 2021, Austin, Texas, USA).

The results of the dissertation were also utilized in research work supported by Russian Science Foundation (RSF) grant 21-19-00516 "Multi-Agent Adaptive Control in Networked Dynamic Systems Applied to Groups of Robotic Devices under Uncertainties" and St. Petersburg University grant ID: 94062114.

Publication of Results. The main results of the research are reflected in the works [12, 13, 73, 78, 88–90]. The applicant has published 8 scientific papers, of which one publication is a computer program registration certificate, six published in publications indexed in the Scopus database, and two in a journals included in the list of peer-reviewed scientific journals in which the main scientific results of dissertations for the degree of candidate of sciences should be published.

[12,13,73,78,88–90] are co-authored. In [73] S.Yu. Sartasov was responsible for the methodological and organizational support of the work, review of sources and analysis of the results, co-authors — for an independent review of sources. In [89] S.Yu. Sartasov provided general statement of the problem, the architecture of the solution and methodological support, the co-authors worked on the choice of solution methods and its programming. In the works [12, 13, 88] O.N. Granichin provided general methodological guidance, S.Yu. Sartasov contributed to general statement of the problem, architecture of the medium risk functional and the solution as a whole, testing methodology, processing of experimental results, other co-authors created a test bench, provided more details to the medium risk functional, programmed the solution and test scenarios and conducted experiments. In [90] S.Yu. Sartasov provided general statement of the problem, the choice of solution methods, the architecture of the solution, the processing of the results of experiments, the co-authors created test benches, programmed the solution and conducted experiments. In [78] O.N. Granichin provided general methodological guidance, S.Yu. Sartasov provided general statement of the problem, development of the medium risk functional, solution programming, testing methodology, conducting experiments and processing their results, M.A. Pelogeiko contributed to modifications of the medium risk functional, programming of the solution, preparation of a test bench and experiments.

The program modules developed based on the results of the dissertation research received an implementation acts by Lanit-Tercom LLC and federal state budgetary educational institution of higher education "Baltic State Technical University "VOENMEH" D.F. Ustinov" (see Appendix A) and a certificate of state registration of the computer program № 2023666564 dated August 2, 2023 [91] (see Appendix B).

Structure and volume of the dissertation. The dissertation consists of an introduction, three chapters, a conclusion, a list of references, including 110 sources. The text occupies 90 pages, contains 3 figures and 12 tables.

Brief content of the work.

Introduction substantiates the relevance of the dissertation research, formulates the goal, sets the research objectives, and summarizes the main results.

First chapter describes the problem of measuring CPU power consumption, followed by a literature review. Section 1.1 outlines the concept of energy consumption of mobile devices, defines the problem of optimizing energy consumption, and considers existing approaches to solving this problem both at the hardware level and at the level of operating systems and individual programs and software systems. It is separately noted that the confirmation of the quality of the solution is the comparison of the energy consumption of the device or its component before and after its implementation. Section 1.2 discusses the methodology for conducting measurement experiments and provides an overview of existing approaches to measuring energy consumption. The difference between measuring and estimating energy consumption is established. Existing approaches for CPU power consumption, their advantages and disadvantages are described, and a conclusion is made about the advisability of creating a new evaluation model. Section 1.3 describes and substantiates a new CPU power consumption model that takes into account the peculiarities imposed by a heterogeneous architecture and uses the author's approach to accounting for CPU idle time.

Second chapter explores the applicability of randomized stochastic optimization algorithms to the heterogeneous CPU power management problem. Section 2.1 introduces the basic concepts of the theory of stochastic optimization, describes the most developed algorithms and their properties. Section 2.1.1 defines the medium risk functional and the problem of finding its optimum. Section 2.1.2 describes various variants of the SPSA algorithm and related theoretical results, including validity of estimates. In Section 2.2, the connection between the problem of optimizing processor power consumption and the problem of tracking changes in the optimal parameters of the non-stationary medium risk functional is considered, and theoretical results in this area for the SPSA algorithm are given. Section 2.3 develops the medium risk functional for the SPSA algorithm with one noisy observation. Section 2.4 develops the medium risk functional for the SPSA algorithm with two noisy observations. Subsection 2.4.1 defines the concept of program execution cost and, based on this concept, target function for the SPSA algorithm with two observations is formed. Section 2.4.2 discusses various approaches to obtaining noisy observations within the DVFS cycle.

Third chapter is about designing DVFS governors based on the principles outlined in Chapter 2 and testing them. Section 3.1 describes the technical aspects of programming the SPSA1 and SPSA2 governors for Android OS. Section 3.2 presents the methodology and results of experiments to evaluate the performance of the obtained governors. Subsection 3.2.1 is devoted to testing the SPSA1 governors, subsection 3.2.2 — testing various variants of SPSA2. Section 3.3 analyzes the obtained results and concludes that the properties of the obtained governors satisfy the conditions of the set goal.

Conclusion formulates the main results of the dissertation.

Chapter 1. Heterogeneous Central Processing Unit Power Model

This chapter introduces the basic concepts related to the architecture and power consumption of modern mobile devices, discusses various approaches to optimizing power consumption and how to measure or evaluate it, and formulates a power consumption model for a heterogeneous central processing unit (CPU).

1.1 Energy consumption of mobile devices

Mobile devices such as smartphones, tablets and smartwatches have become an integral part of modern life. Digital services received through these devices not only improve the quality of life, but also form new types of relationships in society. The number of actively used smartphones is expected to exceed 7 billion in 2024 [74]. Android is the most widely used operating system (OS) for mobile devices today.

Mobile devices are electrical devices with an autonomous power source (battery) first and foremost, and therefore they have a limited battery life. From a physical point of view, lithium-polymer batteries used in modern mobile devices are voltage sources. At a charge level above 90% the voltage between the battery contacts is higher than the nominal voltage, from 90% to about 20% the level of voltage produced by the battery changes little, and then begins to decline sharply. Because of this, and due to additional circuits, in practice the battery voltage of a mobile device is assumed to be constant or slightly changing within the framework of the experiment. The main characteristic of a battery is its capacity, measured in ampere-hours (A·h). For example, a capacity of 4000 mA·h means that the battery can provide a current of 4 A at its nominal voltage for one hour, while increasing or decreasing the current proportionally reduces or increases the operating time [10].

Different modes of operation of mobile device components such as the CPU, display, Wi-Fi and Bluetooth® wireless modules, graphics card, etc. have different power consumption levels. In particular, the CPU has the ability to switch between operating frequencies, and the lower the selected frequency, the lower the power consumption. The power consumption of the screen is the sum of the costs for the formation of pixels of a certain color and the backlight of the screen. Backlight power consumption grows exponentially with brightness levels, however, the PenTile pixel arrangement for AMOLED displays uses more power when displaying green images compared to red and blue images due to twice the number of green subpixels per pixel [16]. Wireless communication modules load the battery much more heavily in network search and connection establishing modes compared to stable network operation on a finalized configuration, which leads to increased power consumption in a situation of unstable network connectivity [90].

System software and applied software use the components of the device differently, but due to the use of the CPU, any computing load inevitably drains the battery. Thus, the task of increasing the battery life of a mobile device is essentially the task of optimizing power consumption. Additional relevance to the problem is also added by the fact that electrochemical batteries used in mobile devices available on the market degrade over time, that is, their maximum capacity decreases with each recharge cycle [10]. Therefore, the additional economic effect of solving this problem arises not only due to a decrease in the energy consumed when recharging mobile devices over time, but also due to the ability to increase the time before buying a new mobile device, for the reason that the battery life of the old one has decreased to unacceptable level.

On the other hand, from a user point of view, it is important to maintain the comfort of using the device, regardless of the chosen energy management strategy. Since a decrease in power consumption is associated with a decrease in performance, then, informally speaking, the goal of optimization is to make the device to consume as little energy as possible while remaining as comfortable as possible for the user. However, due to the fact that the concept of "comfort" cannot be objectively formalized, within the framework of various approaches to solve the problem more formal optimization criteria are created, which, according to their authors, are associated with this concept.

The classic approach to solve the problem of increasing battery life is to physically increase the capacity of electrochemical batteries, and significant progress has been made in this direction over the past 25 years: nickel-cadmium batteries used in handheld computers in the early 2000s had a capacity of about 1000 mA *cdot*h, while the typical capacity of lithium-polymer batteries in modern smartphones is 3500-4000 mA·h [10]. Another widely used scheme is to turn off or put to sleep (hibernate) a component of a mobile device if it is not used for a long time — in this state, its power consumption is considerably less. The result of this approach is most often noticeable when, after some time, the screen of an unused mobile device turns off, but the drivers for both wireless modules and graphics cards still work [3].

At the applied software level, energy optimization is achieved by introducing energy profiles into applications — sets of configuration values that are applied at a certain charge level [86]. For example, when the charge level reaches 20%, the application starts using the geopositioning module (GPS) once every 5-60 seconds instead of constantly. To conserve battery power, dark UI themes may be used instead of light UI themes [24]. A separate area of research is the processing of computing tasks not on the device itself, but in the cloud infrastructure with the transfer of results back to the device — the so called offloading scheme [20, 107], which is complemented by the spread of the serverless application architecture [77].

A significant attention is paid to energy consumption issues in the system software. If a mobile device component has technical capabilities to optimize power consumption, such capabilities are actively used at the operating system level.

Modern CPUs are capable of entering an idle state, during which the execution of any programs is suspended. Multi-core processors can put one or more of their cores into an idle state while other cores remain active. In the idle state, power consumption is reduced by turning off some of the processor hardware components. The deeper the state, the more components are disabled, however, this increases the time it takes to get back to the active state. In terms of Linux and Android OS, the total delay to enter an idle state and the minimum time that the hardware will be in it is called target residency.

For example, a dual-core ARM CortexA9 processor has the following idle states, in order of increasing depth [69]:

- C0 – active state.
- C1 – most processor timers are off. Exit delay is 4 µs.
- C2 — CPU is turned off, Memory Protection Unit (MPU) is turned on to protect critical data, core is inactive. Exit delay is 1100 µs.
- C3 - analogous to C2, but the core is in CSWR mode. Exit delay is 1200 мкс.
- C4 - analogous to C3, but the core is in OSWR mode. Exit delay is 1500 мкс.

Usage of CPU technical ability to enter a certain idle state depends on the mobile system-on-chip and OS, therefore there are mobile devices available on the market that can only enter the C1 state, while the installed CPU supports deeper idle states.

An idle state governor is a module in the Android OS kernel that can monitor the current state of the system and send a signal to the CPU to enter or exit some idle state for one or more of its cores. There are several generic controllers available on Android devices and the default algorithm is menu. This governor tries to predict the current idle duration, then adjusts the resulting value based on a number of factors, and then tries to find the deepest idle state given the target residency and exit delay. The current idle length prediction is based on the history of previous idle length.

The installation of multi-core processors on mobile devices made it possible to implement true multitasking in mobile operating systems. Many multi-core processors have a homogeneous architecture, that is, the processing capabilities of each core are the same. Current OS task schedulers that support this CPU architecture do not include scheduling disciplines aimed at optimizing power consumption. These include the Completely Fair Scheduler (CFS). The situation has changed significantly with the advent of multi-core CPUs with a heterogeneous architecture. An example of such an architecture is big.LITTLE from ARM [45]. All present cores are divided into several clusters, and the cores are homogeneous only within their cluster. Therefore, it becomes possible to run programs that are undemanding to computing resources on weaker, but energy-efficient cores (the so-called "LITTLE"), while more computationally powerful and energy draining cores ("big") are used for high priority or resource-intensive tasks.

Energy-aware scheduling (EAS) [54] is a task scheduler that can only work properly on systems with heterogeneous CPUs. It was developed for Linux OS but is also supported in Android. EAS uses normalized energy models for the cores of each cluster, taking into account the available frequencies and power consumption. It is sufficient to simulate the energy behavior for one core of the cluster, since the cores inside the cluster are homogeneous, and in real systems-on-a-chip, power supply and frequency control is carried out for all cores of the cluster simultaneously and unified. When a task needs to be scheduled, EAS chooses the core to run that task in such a way that the CPU power consumption increases as little as possible. EAS runs as long as CPU usage is below 80%, otherwise CFS is used until the load is reduced. Unlike classical schedulers, in order to achieve power savings, EAS must be able to give control signals to CPU clusters, which is done in other OS modules that are also considered part of EAS.

One of the most developed approaches is Dynamic Voltage Frequency Scaling (DVFS) [103]. The CPU is capable of operating not at a single frequency, but in a certain range. This range is limited from below by considerations of user convenience, and from above by issues with heat dissipation, since heat generation also increases with increasing frequency. Since the existing heat dissipation mechanisms are limited both in terms of power consumption and size, the main way to cool the processor of a mobile device is air cooling. At the time of writing, the limit of the frequency range in which heat dissipation can be effectively dissipated by air cooling is about 2 GHz for models available on the market. Although in general the power dissipated by the processor is proportional to the cube of the operating frequency, in the discussed range the relationship between power consumption and frequency can still be approximated by a linear function.

The DVFS governor is an algorithm or an OS module that implements it[1] that monitors the current state of the device and sends CPU signals to increase or decrease the operating frequency according to some strategy. It should be noted that the signal of the DVFS governor is advisory, and the CPU can respond to this signal with a noticeable delay from the point of view of the governor.

There are a number of publicly available DVFS governors for Android OS. Some of them are specific to this OS, while others were originally used in the Linux kernel [23]:

- Powersave. This governor sets and holds the lowest available frequency. It saves the most energy but delivers the worst performance.

[1]Hereinafter, in the context of theoretical analysis, the DVFS governor will be understood as an algorithm, and in the context of practical implementation — an OS module

- Performance. This governor works exactly the opposite: it sets the maximum frequency for best performance at the cost of maximum power consumption. Performance and PowerSave correspond to two extreme optimization strategies and are often used as reference algorithms for comparison.

- OnDemand. This governor sets the CPU frequency in proportion to the maximum load, defined as the ratio of active CPU time to total running time among the CPU cores observed between governor calls, allowing the device to respond to changes in computational activity. When a certain CPU load threshold is reached (default is 80%), the maximum frequency is set until the load drops below the threshold again.

- Conservative. This governor is a continuation on OnDemand ideas: it increases the frequency when there is activity on the CPU, and reduces to a minimum gradually, rather than in spikes typical for OnDemand.

- Interactive. This governor is designed specifically for the Android OS and takes into account the fact that if the user starts interacting with the device, he will continue to do so for some time. The operating frequency still depends on the load level, but this level is estimated not only after a fixed time has elapsed, but also when certain hardware interruptions occur, which include those resulting from user interaction with the device, for example, from touching the screen.

- schedutil. This governor is designed to work exclusively with the EAS task manager. Its basic idea is the same as in OnDemand, but the load definition is based on the EAS energy model, rather than the ratio of active time to total time.

There are also conceptually different approaches to this issue.

Unlike EAS, which uses a static energy model, AdaMD implements a scheduling routine that regularly checks the status of the various device resources required by currently running processes and reassigns them to more appropriate cores as needed [9]. Compared to other approaches that bind threads to cores, this method saves up to 28% of energy while meeting 95% of performance requirements.

In recent years, the application of neural networks for building general-purpose DVFS governors has been actively researched. For example, the "long short memory" effect of a recursive network can be used to avoid the gradient damping problem, since control signals should only be generated for data received within a recent time [60]. This network architecture reduces processor power consumption by up to 19% compared to standard DVFS algorithms.

The DVFS governor could be built using other optimization criteria than performance or energy efficiency. The temperature of the CPU chip is another important factor, as an increase in temperature leads to an increase in power consumption. By using a neural network in the DVFS loop, the average chip temperature can be reduced by 18 °C with minimal execution overhead and comparable performance [83]. Chip temperature can be combined with power efficiency as an optimization criterion, and using a deep learning neural network with reinforcement to determine the CPU temperature and estimate the ambient temperature can result in a 23.9% reduction in power consumption while maintaining the required level of performance [58].

DVFS algorithms can also be built to be used with specific applications, rather than for the general case. For example, a DVFS governor for augmented reality applications that takes into account frame rate and response time requirements can theoretically reduce power consumption by 80%, but this has not yet been experimentally confirmed [93].

DVFS is also used for the Graphical Processing Unit (GPU), which is used to offload a number of computational tasks, most often associated with graphical algorithms, from the CPU. It can also operate at different frequencies, so there exist a concept of a single DVFS algorithm for both devices. The rules-based model can reduce power consumption by 18.11%, and the frame rate of the smartphone is reduced by 3.1% [75]. Another approach is to combine load, energy, and temperature data for the CPU, GPU, and operating memory, and assign a frequency to each component using a common priority list. This method saves at least 26.8% of energy compared to default regulators and modern approaches [25].

Existing approaches are not without certain drawbacks. To begin with, it is worth noting that the adaptation of existing DVFS governors, including classical ones, to work on a heterogeneous CPU architecture is done by interpreting each homogeneous cluster as a separate multi-core processor. Due to the circuitry features of both crystals and systems-on-a-chip, the same operating frequency is set for all cores within a single cluster. Thus, in theory, the result of the governor operation at each iteration of the DVFS cycle is the operating frequency from the list of frequencies available for the cluster, at which it will operate until the next iteration, where it may change. In practice, among the considered governors, none of them takes into account the fact that the cluster can switch the frequency for a longer time than the duration of one iteration of the DVFS cycle, which makes certain iterations of DVFS loop meaningless. For example, it was found during this study that although the typical nominal DVFS cycle length in Android OS is 10ms, the actual switching time of Helio G90T CPU clusters can reach 30ms. Thus, upon a successful request to switch the frequency at some iteration of the DVFS cycle, the results of up to two subsequent DVFS cycles are ignored. From a technical point of view, it would be useful to take this fact into account in the work of governors.

Another problem is the fact that classical algorithms like OnDemand, Interactive and EAS+schedutil optimize power consumption based on empirical observations and models, including the load change model. The main goal of the dissertation research was to build a DVFS governor based on the principles of operation with proven validity of estimates to the optimal value under uncertainty of load from one iteration of the DVFS cycle to the next.

1.2 Measurement and evaluation of energy consumption, methodology of experiments

Methodologically, the comparison of two DVFS approaches in terms of power consumption optimization is done as follows: on the same device, which is in the same state before the start of the experiment, the same sequence of actions is performed in order and in time, while the OS processes are launched at the same time and the trace of their execution is the same. After that, data is collected about the power consumption of the device or its individual component during the passage of the sequence of actions. Such an experimental methodology requires the solution of a number of issues.

First of all, it is impossible to achieve a perfect match between the state of the device before the start of experiments and during operation due to the inability to control a number of factors. For example, triggering a hardware interrupt can affect the distribution of tasks between cores by the OS task scheduler. In the literature, there is a vision of how to minimize the influence of external factors before and during the experiment [73]:

- Using automated tests instead of manual ones [22].
- Reducing the activity of background processes (both system and non-system) [15, 46, 53, 76, 85, 102].
- Increase the system priority of the application that is the target of the experiment [46].
- Reduce screen brightness or turn it off completely [15, 47, 49, 52, 76, 85, 98].
- Disable all unused device components (Wi-Fi, 4G, GPS, accelerometer, range finder, etc.) [1, 15, 53, 76, 87].
- Charging the battery to the same level [8, 15, 98].

- Turning off the device to cool the battery [15].
- Warming up by executing tests without measuring power consumption [72].
- Waiting before starting a new test to end or reduce the activity of background processes [28, 85].
- Reinstalling the application under scrutiny [47].
- Clearing the cache of the apps in question [26, 85].
- Reboot device [87].
- Factory reset [8, 87].

Nevertheless, it is impossible to completely avoid the influence of all external factors on the conduct of the experiment, so they are carried out several times and the measurement results are statistically processed.

From an experimental point of view, making a device to use a different DVFS algorithm is well isolated from other factors due to the organization of the DVFS system in the Android OS.

A much less straightforward issue is how to measure the power consumption of a device or its individual component. From a conceptual point of view, existing approaches can be divided into two classes:

- **Direct measurement approaches** directly measure the energy consumption of a device or component using special equipment.
- **Indirect measurement approaches** measure some secondary metrics related to the energy performance of the device and estimate the resulting energy consumption based on them. Indirect methods are always based on some model that relates the measured metrics and the resulting energy consumption, so this class of approaches can also be called model-based.

In **direct measurement approaches** equipment is connected to the target device to measure voltage, current, or power. Such approaches work with physical devices and fundamentally cannot work with emulators. Within this class of approach, the following can be distinguished:

External measuring device: A digital multimeter is connected to the battery contacts of the device or to the power circuits of the component of interest, which is much more complicated. Sometimes, in order to compensate for the voltage drop in lithium polymer batteries during discharging and, therefore, to normalize the power values, instead of the battery, the device under test is powered from an external voltage source. Within the framework of the scenario (script) of the experiment, both the considered application or unit test and the recording of power measurements are launched. Since the multimeter works in discrete time and averages the current or power over the past cycle, the total power consumption can be estimated using linear interpolation:

$$E = \sum_{i=1}^{N_{read}-1} U_i \cdot \frac{I_{i+1} + I_i}{2} \cdot (t_{i+1} - t_i),$$

where E — total energy, N_{read} — number of multimeter measurements, U_i — i-th voltage measurement, I_i — i-th current measurement, t_i — timestamp of the i-th measurement. Some multimeters can record the timestamp directly, for others the difference $t_{i+1} - t_i$ can be defined as the reciprocal of the multimeter frequency. Step interpolation can be used instead of linear interpolation.

If a more detailed measurement is needed, for example, at the level of energy consumption of individual program methods or code blocks, then the source code is supplemented with instructions for logging the beginning and end of the method body or code block (instrumented), and two data traces are recorded — power readings and an execution trace applications, as described by Couto et al. [22]. In this case, before starting experiments, the system clock on the device and the multimeter or control computer must be synchronized. The power readings can then be compared to the execution trace to get an idea of the power consumption of a method or block of code. Since instrumentation of the application requires additional program code, the energy costs of its execution must be estimated and subtracted from the final result [15, 21, 27, 47, 57, 59, 63, 97, 105].

Internal measuring device: This approach uses the internal power meters installed on the device by the manufacturer and the Android OS API to access them. Often the configuration of the measuring device is such that the energy consumption of the device as a whole is measured, so the measurement of the energy consumption of an individual component requires additional methodological processing of the experiment. An external voltage source can also be used instead of a battery. Since the device and the power sensor use the same system clock, there are no time synchronization issues. Requests for power readings may be part of the instrumentation code. The methodology for obtaining the total value of consumed energy does not differ from that for the external meter approach [18, 31, 51, 53, 59, 102, 105].

It should be noted that the listed strategies are aimed at measuring the energy spent by the device or individual components, but changing the battery level allows you to estimate the amount of energy given to the device, especially since such sensors are generally available on mobile devices. In practice, measuring the battery charge level is less preferable than measuring the energy spent, for the following reasons:

- Accuracy of estimation of charge sensors up to tenths of a percent is insufficient for experiments lasting seconds and minutes.
- Each battery charge cycle reduces its actual capacity, so a change in charge by 1% can mean a significantly different amount of energy delivered even within the same series of experiments. The initial and subsequent determinations of the actual battery capacity require additional labor.

Summing up direct measurement approaches, it should be said that their main advantage is exactly the measurement of the energy consumption of a particular physical device, and not an estimate. The main disadvantages are the financial cost to the experimenters and the higher skill level required to build the test bench[2]. In addition, the methodology of experiments within the framework of this approach should take into account a number of features of working both with multimeters and with the devices themselves:

1. Even within the same device model, different devices exemplars may have different power consumption. Although in the documented cases it did not affect the internal consistency of the results of experiments on one device, nor the general conclusions for all devices, this risk should potentially be taken into account [110].

2. Connecting measuring tools to a device is very difficult if you need to measure the power consumption of a single component, — up to the use of specialized boards for testing, for example, Odroid-A [92], instead of a real device. Since the hardware configuration of such boards is made similar to mobile devices available on the market, and they use typical versions of mobile operating systems, such a device replacement for testing is legitimate.

3. The operating frequencies of modern processors and peripherals are in the gigahertz range. However, the measuring instruments used in the literature operated at much lower frequencies. The maximum frequency of 100 kHz is documented by Wilke et al. [105,106], other works use multimeters operating below 10 kHz. Because multimeters average current between clock pulses, shorter duration power peaks can be smoothed out to levels of measurement error and go unnoticed. The following methods of taking this phenomenon into account in the experimental methodology are known in the literature:

   - Mittal et al. [71], Dolezal and Becvar [28], Saksonov [87], Pandiyan

[2]Although no experimental devices were damaged during this study due to improper assembly of the test bench, the risks of such a development and subsequent material loss were quite real.

and Wu [76] disabled DVFS or set the operating frequency to a constant level, thus making the CPU power consumption more stable. This method is not suitable if the purpose of the experiment is to compare the power consumption of a device controlled by different DVFS modules.

- Hung et al. [50], Yoon et al. [109], Zhang et al. [110] include CPU frequency data as additional input into a formula for estimating total energy consumption.
- Couto et al. [22], Li and Gallagher [64] set the test run time so that the duration of the test will average out the power consumption estimate even when using low-frequency equipment.
- Larsson and Stigelid [59] adjust how often the internal multimeter is accessed for power consumption data depending on the duration of the test. According to the results of this study, the use of this approach is recognized as counterproductive and is not recommended, since for Wi-Fi and Bluetooth modules, an exponential dependence of the recorded power consumption of an electronic component on the frequency of accessing the internal multimeter was experimentally shown. Nevertheless, after calibration, one can enter a correction factor that will eliminate the influence of this phenomena [90].

Indirect measurement approach (or model-based approach) assumes that the profiling software collects some information about code execution and relates it to energy consumption through a mathematical model. The work is carried out in two stages: model calibration and energy consumption estimation. In the first step, the coefficients of the model are determined or adjusted using auxiliary experiments or reference data. As mentioned above, this is important not only for different device models, but also for different devices of the same model [47]. Once the model is configured for the device, it can be used to estimate power consumption based on indirect data obtained during the experiments.

Depending on the type of input data, models can be classified as follows:

Working time models: In this approach, the result of experiments is information about the operating time of various components of the device [7,8, 22,26,28,32,49,50,52,57,61,62,68,71,85,87,92,98,109,110]. Different operating modes may have different power consumption profiles. For example, the Wi-Fi module consumes a different amount of power in standby mode, when searching for a network, and when transmitting data over the network, and in the case of the screen, different levels of power consumption at different levels of backlight intensity could be considered as such profiles. The energy consumption estimate is calculated as

$$E = \sum_{i=1}^{N_{dev}} \sum_{j=1}^{N_{P_i}} P_{ij} \cdot t_{ij},$$

where E — total energy, N_{dev} — number of smartphone components included in the assessment, N_{P_i} — number of energy consumption profiles of the i-th device, P_{ij} — average value of power consumption for j-th operating mode of i-th device, t_{ij} — total operating time of i-th device in j-th operating mode.

During the calibration phase, experiments are carried out to determine the energy profiles for individual components using one of the direct measurement approaches. From a conceptual point of view, both external and internal measuring instruments are suitable. Linear regression is applied to extract the power parameters used as weights. At the same time, device manufacturers themselves can indicate data on the power consumption of devices in a special file of power profiles (power profile) of the Android OS [82].

To estimate power consumption, the experiments measure the active time of each device of interest during the execution of the test load. Certain technical inconveniences arise due to the fact that various components report their operating time differently, for example, in the Android OS, the CPU stores information about the time spent at various operating frequencies in the `proc` folder, and the Wi-Fi module generates system events when switching from one energy profile associated with the current state of the component to another. Additional correction factors can also be included in the model, such as corrections for the battery discharge rate [92].

Instruction energy consumption models: This group of models estimates power consumption by correlating individual instructions or groups of CPU instructions — conditional statements, loop control instructions, method calls, integer operations, floating point operations, etc. — and the energy spent on their execution [2,46,64]. Energy consumption is calculated as

$$E = \sum_{i=1}^{N_{instr}} P_i \cdot n_i,$$

where E — total energy, N_{instr} — number of different instructions or groups of instructions in the model, P_i — power consumption of one i-th instruction or instruction from i-th group, n_i —– number of the i-th instruction or instructions from the i-th group in the executed code.

Model calibration is performed by measuring the power consumption of each instruction type in synthetic benchmarks using a direct approach. The total energy is calculated based on the statistics of instructions in the execution trace of the experiment code. It should be noted that the actual launch of the test code under the Android OS is not required if it does not require a call to the Android API. Instruction statistics may be aggregated in any suitable environment.

Energy models for method or API calls: This approach is similar to the previous one, but instead of a power profile for a single instruction, the power consumption of a system call, an API call, or a software framework method is calculated [30,48,72,104]. Models in this approach are built on the assumption that most of the time and energy is spent outside the application code, and so one can get a pretty good estimate of the application power consumption by analyzing its API usage.

Summarizing the various options within the indirect measurement approach, it should be noted that in practice, working time models are used more often than other types of models. On the one hand, this is due to the fact that the time spent by a device component in any state is quite simple to register, and the Android OS has already implemented the corresponding API for various components. On the other hand, the power consumption models of instructions and methods do not take into account all the nuances of the CPU architecture, for example, pipelining, shared caches between several CPU cores, preemptive multitasking hardware technologies, for example, Intel HyperThreading [66], etc., and therefore their accuracy should be treated with caution. However, the argument about the accuracy of model calibration is generally valid for the entire approach, since any model relies in one way or another on direct measurements. The undoubted advantage of the approach is its cost and lower demands on the qualification of experimenters.

Google, which supports the Android operating system, recommends using the working time model to estimate CPU power consumption [67]. According to this model, the CPU, operating at a certain frequency, consumes a constant (averaged) current at a constant voltage. Therefore, the total energy is estimated as

$$E = U \sum_{i=1}^{n_f} I(f_i) \cdot t_{f_i}, \tag{1.1}$$

where n_f is the number of frequencies available to the CPU, f_i is a specific frequency, $I(f)i)$ is the current consumed at that frequency, t_{f_i} is total time the CPU has been running at f_i. Moreover, the equation can easily be extended to a heterogeneous architecture:

$$E = U \sum_{c=1}^{n_c} \sum_{i=1}^{n_c} I(f_{ci}) \cdot t_{f_{ci}}, \tag{1.2}$$

where n_c is the number of heterogeneous architecture clusters, and frequencies and times are taken into account for each cluster separately.

Table 1.1: 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

The data for the equation (1.2) is available in Android OS. In particular, the $t_{f_{ci}}$ values for each core are stored in special `time-in-state` files in the `/sys` directory. Time data is stored line by line for each frequency in the format "`<frequency><time>`". The number of rows is $n_c f$. Since CPU clusters are registered within OS as separate processors, the composition and content of lines in `time-in-state` files varies from core to core, depending on the clusters they belong to. Time is measured in 10 ms increments and starts from the moment the corresponding OS driver is started or restarted. The weights $I(f_{ci})$ are stored in the power profile file `power_profile.xml` in mA, as mentioned above. According to the Google specification, this file must be included by the manufacturer within the Android OS build on the device, but, unfortunately, manufacturers often neglect to do so [89]. However, if the file does not exist on the selected device, constants from other devices with the same CPU model and the same or similar cluster topology will do. An example of the contents of such a file for the CPU is presented in Tab. 1.1.

In general, the estimate of power consumption in mA·h is not complete due to the non-constant voltage. But since it is assumed that in the case under consideration the voltage on the CPU is constant, this constant factor is neglected. Moreover, the nominal voltage of mobile CPUs in the current market is 1 V, so the estimate obtained from this model in mA·h and converted to A·s, in practical terms, turns out to be numerically equal to the estimated power consumption in J.

1.3 Building a power consumption model for a heterogeneous CPU

As noted above, although Google's basic power consumption model finds its way into research papers, it does not take into account time when the CPU is idle. Moreover, a study of the source code of the Linux and Android OS kernels showed that the contents of the `time-in-state` files are formed without taking into account the operation of the idle state governors. To solve this problem, the following modification of the basic model is proposed.

The (1.2) equation implicitly assumes that the CPU, being at a certain frequency, consumes constant power, regardless of whether it is actively computing or not. This assumption is confirmed by the implementation of the so-called idle task. When the OS scheduler fails to schedule an active task for the core, but the core is in the C0 (active) state, the scheduler assigns a special idle task that consists of a sequence of NOP (No Operation) instructions. However, the idle state C1 already turns off the core timers, and no computational task can be processed on it. Considering that only C0 and C1 states were available on all devices used in this study, and the fact that the modern implementation of a heterogeneous architecture implies the presence of several clusters, all cores within which operate at the same operating frequency, the equation (1.2) can be changed like this:

$$E = \sum_{i=1}^{N_{cl}} E_{idle_i}(t) + \sum_{i=1}^{Ncl} U_i \cdot \sum_{j=1}^{N_{f_i}} I_i(f_j) \cdot \sum_{k=1}^{N_{cores_i}} t_{f_{jk}}, \tag{1.3}$$

where N_{cl} — number of core clusters, $E_{idle_i}(t)$ — energy consumed by the enabled CPU cluster in the absence of active computing processes, N_{f_i} — number of available frequencies in the i-th cluster, U_i — nominal voltage of the cores in the i-th cluster, $I_i(f)$ — nominal current for the core in the i-th cluster operating on selected frequency f, N_{cores_i} — number of cores in i-th cluster, $t_{f_{jk}}$ — time spent by k-th core i- th cluster at the jth frequency.

It can be seen that the proposed model differs from Google's base model in that instead of cluster-wide power constants, individual core power constants are used. The justification for the legitimacy of such a replacement is the following experiment with the Samsung Galaxy A3 (2016) smartphone, which has an Exynos 7578 processor with 4 Cortex-A53 cores and a maximum clock frequency of 1.5 GHz [88]. The battery power was changed to a 3.85 V DC supply and an ammeter was added to the circuit. Cores 2 through 4 were disabled using Android OS system calls, while the first only handled background processes. Peripherals have been disabled or put into power saving mode. Then, an infinite computational cycle was launched on the first core (decomposition of an infinite sequence of natural numbers into prime factors by brute force). The thread executing the program was pinned using Android OS system calls to run on the first core only. Then, without stopping the first thread, the second core was turned on, and the same program was executed with pinning to it. So the cores were switched on sequentially up to the fourth and the current was measured. The experiment was run in two stages — with the Performance governor and the Powersave governor to maximize and minimize performance and power consumption. The measurement results are shown in Fig. 1.1.

It is worth noting that point 0 corresponds to the basic power consumption of the entire device at rest and the flow of background processes. Such

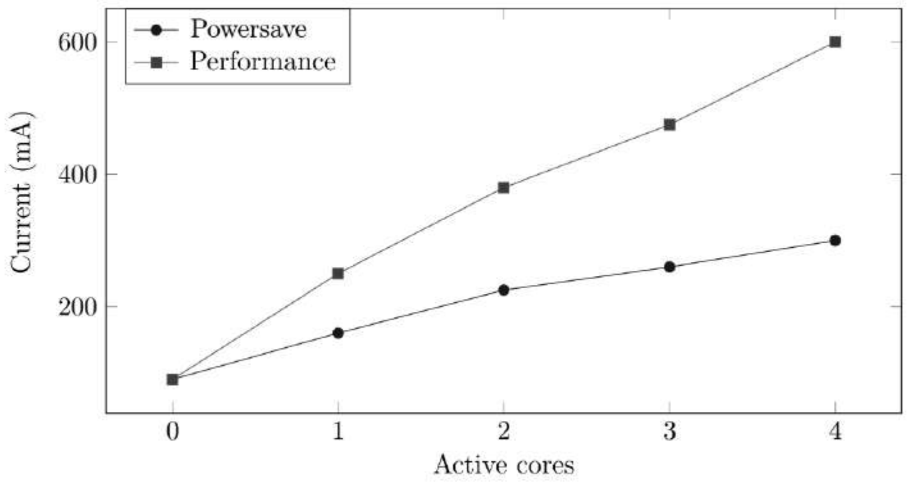


Figure 1.1: Power consumption of the Samsung Galaxy A3 (2016) smartphone depending on the number of active cores

a power consumption graph was obtained for any order in which the cores were turned on. It can be seen from the graph that although the dependence of power consumption on the number of active cores is not strictly linear, from an empirical point of view, it is quite close. Therefore, in this study, the transition from cluster-level power consumption constants to single-core level power consumption constants is considered justified and, in the absence of more detailed information about the operation of the cluster, is made by simply dividing the corresponding cluster constant by the number of cores in it.

This model is more complicated than the model described in (1.2), but for a number of practical reasons it can be simplified. The first term assumes that the idle power consumption of each cluster is linearly dependent on the operating time. Although it is logical to assume that for a more accurate calculation of idle state power consumption, it is necessary to divide $E_{idle_i}(t)$ into two parts (baseline CPU power consumption and C1 state power consumption for each core), due to the lack of information about the baseline CPU power consumption profile and about the C1 profile both in `power_profile.xml` and in the publicly available CPU and system-on-a-chip specifications for the devices used in this study, we omit this term from further analysis.

Next, evaluation of

$$\sum_{k=1}^{Ncores_i} t_{f_{jk}}$$

seems redundant, since the cores of the same cluster always operate at the same frequency. However, this term is necessary when we are trying to evaluate the effect of deep idle states on power consumption.

Although the idle state stops the CPU core timers, it does not change the operating frequency of the core itself, i.e. a core running at 1 GHz before entering the idle state will continue to operate at 1 GHz after exiting the idle state, if the operating frequency the entire cluster has not changed during this time. The original assumption was that the idle governors put the core into one of the idle states, when there are so few tasks that the scheduler cannot load all the available cores, and in such a situation the DVFS governor sets the minimum operating frequency for the cluster. Therefore, the original idea of the model was to subtract the idle time in C1 from the time spent by the core at the lowest frequency:

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

where t_k^{idle} is the time spent by the k-th core in the idle state. However, this assumption was not confirmed, since during the experiments situations were found when $t_{f0k} < t_k^{idle}$. This means that the OS put the kernel to sleep at a frequency other than the minimum. This phenomenon can be explained by the fact that under specific conditions, when switching the operating frequency of the cluster can take even longer than the length of the target residence, and the load on an individual core during the last DVFS cycle can be 100%, a number of DVFS regulators may not have reasons to lower the frequency for the cluster to the minimum. For example, this is how the DVFS OnDemand governor and the menu idle governor can interact. OnDemand controls the frequency of the entire cluster, so its estimate of the cluster load is defined as the majorant of the load of all cores of the cluster, which means that in a situation where only one core is loaded, the governor will set a high frequency for the entire cluster. At the same time, menu can put low load cores to sleep.

While it is technically possible to modify the Android OS kernel and keep track of what frequency the kernel has been suspended at, this is quite expensive. Each individual build of the Android OS may require to modify the same functionality in different ways. To implement a simpler approach, it will be assumed that idle state management does not depend on the operation of the DVFS governors, but depends on the operation of the OS task scheduler, and the kernel can be put into sleep mode at any frequency at any time. As part of the modified power consumption model, it is proposed to subtract from each individual time spent by the core at a particular frequency such a share of the total idle time that this time at a frequency is from the total operating time:

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

This approach is not precise, but gives a good estimate of actual power consumption. However, data on the idle time is easy to obtain from the Android OS in the same way as information on the time of operation of the cores at a certain frequency. Since this model was developed in the second half of the study, both the original base model and the modified model are used to present the results.

Chapter 2. Randomized stochastic optimization algorithms in the problem of optimizing the power consumption of a mobile device

This chapter provides an overview of randomized stochastic optimization algorithms and shows their applicability to the design of energy efficient DVFS governors. The concept of the cost of program execution is introduced, and for various variants of the algorithm of simultaneously perturbed stochastic approximation, target functionals of the average risk are introduced.

2.1 Randomized algorithms of stochastic optimization

2.1.1 Medium risk functional

In many control problems, the behavior of the target system can be described in an empirical quality functional, the averaging of which determines the *medium risk functional*. Based on the extrema of this functional, the choice of optimal control action is carried out.

More formally, let the probability distribution $P_w(\cdot)$ generate a sequence of p-dimensional random vectors $\{w_n\}$ from $\mathbb{R}^p$. Then finding the minimum of the function $f(\cdot)$ minimizes the mean risk functional if the function has the form

$$f(x) = E_w F(x, w) = \int_{\mathbb{R}^p} F(x, w) P_w(dw),$$

where $F(x, w) : \mathbb{R}^q \times \mathbb{R}^p \to \mathbb{R}$ — loss function differentiable with respect to the first argument, $E_w\{\cdot\}$ — expected value of $P_w(\cdot)$.

When the distribution function $P_w(\cdot)$ is unknown, the problem can be attempted to be solved in cases where it is possible to observe (measure) either values of the function $F(x_n, w_n)$, or values of the gradient vector $\nabla_x F(x_n, w_n)$ at given points $\{(x_n, w_n)\}$, possibly with additional noise. It is assumed that the experimenter can set or observe the values of the sequence $\{x_n\}$, and the corresponding values of the sequence of random variables $\{w_n\}$ from $\mathbb{R}^p$ are generated by the same distribution $P_w(\cdot)$. In this case, the values of $\{w_n\}$ may be unknown to the experimenter, or he cannot influence them.

Measurements of the values of the function $F(x_n, w_n)$ in general case can be made with some additive error $v_n \in \mathbb{R}$ (observation noise). If the disturbances v are centered and independent, then the vector w can be extended with an additional component v:

$$\bar{w} = \begin{pmatrix} w \\ v \end{pmatrix}.$$

Then, instead of $F(x, w)$, we can consider a new function $\bar{F}(x, \bar{w}) = F(x, w) + v$, whose observation scheme will not include additional perturbations, but instead of an unknown distribution $P_w(\cdot)$ it will feature a new unknown distribution $P_{w,v}(\cdot)$. But it is impossible to simplify the scheme of observations in such a way, if the measurement errors do not have "good" statistical properties, [99].

In practice, non-stationary formulations of problems are relevant, in which the minimum point of the functional $f(x)$ and the functional itself change with time. In particular, the problem of controlling the energy consumption of a heterogeneous CPU can be formulated in terms of tracking the minimum of the non-stationary medium risk functional based on incoming observation (measurement) data.

Let's define the minimum point $f_t(x)$ as:

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

Then, to find it, it is necessary to construct a sequence of estimates $\{\hat{\theta}_n\}$ such that $||\hat{\theta}_n - \theta_t|| \to \min$, based on noisy observations of random variables $F_t(x_n, w_n)$, n = 1, 2,.. [44]. In other words, the sequence of estimates $\{\hat{\theta}_n\}$ of the unknown vector θ minimizes the function

$$f(x) = E_w\{F(x,w)\} = \int_{\mathbb{R}^p} F(x,w)P_w(dw)$$

of the medium risk functional type.

Let us assume that the following restrictions are imposed on the function:

Constraint 1: The function $f(\cdot)$ is strongly convex, i.e.

$$\langle x - \theta, \nabla f(x)\rangle \geq \mu\|x - \theta\|_2^2, \forall x \in \mathbb{R}^q.$$

Constraint 2: The gradients of the function $F(\cdot, w)$ satisfy the Lipschitz condition:

$$\|\nabla_x F(x,w) - \nabla_x F(y,w)\|_2 \leq M\|x - \theta\|_2, \forall x, \theta \in \mathbb{R}^q.$$

Although according to the proof of A.T. Vakhitov and O.N. Granichin the boundary can also be formulated with weaker restrictions, from which follow the above, for the purposes of this study they will be sufficient [100].

Next, we consider randomized algorithms that can solve this problem, and then we design target functions that satisfy the constraints of these algorithms.

2.1.2 Simultaneously perturbed stochastic approximation algorithm

Stochastic optimization methods appeared in the fifties of the XX century, starting from the Robbins-Monroe and Kiefer-Wolfowitz algorithms [56, 84].

J. Kiefer and J. Wolfowitz proposed the following solution to the problem of finding a stationary local minimum or maximum θ of some function $f(\cdot)$, when for each value $x \in \mathbb{R}$ the following observation is available:

$$Y(x) = f(x) + v,$$

where $Y(x)$ is the noisy value of $f(x)$ [56]. Under some additional restrictions, the recurrent sequence

$$\hat{\theta}_n = \hat{\theta}_{n-1} - \alpha_n \frac{Y(\theta_{n-1} + \beta_n) - Y(\theta_{n-1} - \beta_n)}{2\beta_n}$$

converges to the point θ, where $\{\alpha_n\}$ and $\{\beta_n\}$ are some decreasing sequences of numbers that must have certain properties. It can be seen that the specified algorithm is actually a kind of pseudo-gradient descent — change of the estimate “on average” goes in the direction of the gradient.

An essential limitation of the Kiefer-Wolfowitz procedure is the implicit assumption about the conditional centering of observation noise, that is, for the function

$$g(x, \beta) = \frac{Y(x + \beta) - Y(x - \beta)}{2\beta},$$

the sample values of which are exactly observed, the mathematical expectation for small β is close to the value of the derivative of the function:

$$E\{g(x, \beta)\} \approx f'(x).$$

This invariant is not always true in practical problems, and because of this, the Kiefer-Wolfowitz procedure does not always converge to the desired point. However, it is also clear from the above definition that the behavior of the sequence of estimates depends on the choice of the function $g(x,\beta)$, and therefore the best behavior of the estimates can be achieved by choosing another function that better approximates "on average" the derivative of $f'(\cdot)$.

Let the function $f(\cdot) : \mathbb{R}^q \to \mathbb{R}$ be twice continuously differentiable, and let $\{\Delta_n\}$ be a sequence of independent random vectors, whose components are independent and have a Bernoulli distribution, i.e. taking the value $+1$ or -1 with equal probability, and at each step n there is no correlation between the values of these random variables and the observation errors. J. Spall proposed to define a new function as

$$\widetilde{g}(x,\beta,\Delta) = g(x,\beta\Delta)$$

and named $\{\Delta_n\}$ as *trial simultaneous perturbation.* He also showed that under the above conditions

$$E\{\widetilde{g}(x,\beta,\Delta)\} = \nabla(x) + \mathcal{O}(\beta).$$

This means that if the condition

$$\lim_{n\to\infty} \beta_n \to 0$$

is satisfied, in the limit this function "on average" coincides with the gradient $\nabla f(\cdot)$, i.e. for large n the probability distribution of appropriately scaled estimation errors is approximately normal. Moreover, the same properties are demonstrated by the function proposed in [34, 79], which uses only one observation at each iteration,

$$\bar{g}(x,\beta,\Delta) = \frac{\Delta}{\beta} Y(x+\beta\Delta).$$

In [34] it is proved that the function $\bar{g}(x, \beta, \Delta)$ allows to construct a sequence of estimates consistent with almost arbitrary noise in the observation.

The question of the rate of convergence of estimates for stochastic approximation algorithms [11, 70, 101] is topical. It was shown that for the Kiefer-Wolfowitz procedure and a number of its generalizations, it depends on the smoothness of the function $f(\cdot)$. For a twice differentiable function, the mean square error of the original Kiefer-Wolfowitz procedure decreases as $\mathcal{O}(n^{-\frac{1}{2}})$, for a three times differentiable one — $\mathcal{O}(n^{-\frac{2}{3}})$. Spall showed that his modification has the same rate of convergence as the original Kiefer-Wolfowitz procedure. Let the characteristic $\gamma = \mathscr{X} + 1$ if all partial derivatives of some function up to order $\mathscr{X}$ inclusive satisfy the Lipschitz condition. [79] suggests using

$$\widetilde{g}_\gamma(x, \beta, \Delta) = \mathcal{K}(\Delta)\frac{Y(x+\beta\Delta) - Y(x-\beta\Delta)}{2\beta}$$

and

$$\bar{g}_\gamma(x, \beta, \Delta) = \frac{1}{\beta} Y(x+\beta\Delta),$$

where $\mathcal{K}(\cdot)$ is a differentiating kernel defined using orthogonal Legendre polynomials of degree less than γ, then for two randomized algorithms obtained using these functions, the root mean squared estimation rate is $\mathcal{O}(n^{-\frac{\gamma-1}{\gamma}})$, and for a wide class of problems it is optimal in some asymptotically minimax sense.

For the multidimensional case, the convergence of stochastic approximation algorithms is proved in [35, 80, 95]. Although the algorithm with one dimension behaves asymptotically worse than the version with two dimensions [79], this is offset by the lower computational complexity of the algorithm [38].

We define *trial simultaneous perturbation* as a sequence of equally 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, $0 < \sigma_\Delta < \infty$. It is convenient to use Bernoulli random vectors as random vectors, since their components Δ_n are independent of each other and take equally probable values ± 1.

There are three possible ways to construct a sequence of estimates of the minimum points of the function $F(x)$ without significant loss of convergence rate:

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

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

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

based on three types of noisy observations with one or two measurements per iteration y_n [40]:

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

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

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

where $\{\alpha_n\}$ and $\{\beta_n\}$ are sequences of non-negative numbers satisfying certain conditions, w_n^+ is the vector of stochastic disturbances for observation y_n^+, v_n^+ is the vector of arbitrary external noise during observations. The third option is convenient in those cases when, due to the specifics of the problem, it is impossible to make two measurements in such a way that they are not correlated with Δ_n [17]. This recurrent procedures are called *simultaneous perturbation stochastic approximation* (SPSA) because they constitutively use a random trial perturbation in all directions. SPSA is a type of stochastic gradient descent algorithm.

The first algorithm uses only one noisy observation, while the second and third use two noisy observations. To distinguish between these variations of the SPSA algorithm, we will refer to the version with one noisy observation as SPSA1, and the last two variations as SPSA2.

Among the conditions for the consistency of estimates, we single out the condition of weak correlation between the trial perturbation $\{\Delta_n\}$ and the sequences of uncertainties $\{w_n\}$ and $\{v_n\}$ as the most important one.

Both variations of the SPSA algorithm work as follows:

1. The target empirical (measured) function $F(x, w)$ is defined.
2. An initial estimate of the optimum point $\hat{\theta}_0$ is made.
3. The current estimate of the optimum is subjected to a random perturbation.
4. The required number of noisy observations of the function F is obtained, and the current estimate of the optimum $\hat{\theta}_n$ is updated.
5. Go to step 3.

It should be noted that SPSA are applicable both for finding a stationary optimum and in problems where the optimum can shift from iteration to iteration, i.e. when solving the tracking problem [42, 43, 96].

2.2 Theoretical results on the properties of estimates of randomized stochastic optimization algorithms in the problem of tracking changes in optimal parameters

In many practical problems, the medium risk functional is non-stationary, that is, its optimum point can shift over time.

In discrete time, the non-stationary mean risk functional can be defined as follows [43]. Let $\{F_\xi(\theta, w)\}_{\xi \in \Xi}$ be the set of differentiable functions $F_\xi(\theta, w) : \mathbb{R}^q \times \mathbb{R}^p$. The set Ξ can be defined as multidimensional, and all further conclusions will be valid for it as well with minor changes, not relevant for this study, but further consideration of the simple case will suffice $\Xi = \mathbb{N}$. Let the sequence $\{x_t\}$ be the observation plan or the sequence of points at which the observer makes measurements at the time moments $t = 1, 2, ...$, and $\{y_t\}$ — sequence of measurement results

$$y_t = F_{\xi_t}(x_t, w_t) + v_t,$$

where $\{\xi_t\}$ and $\{w_t\}$ are sequences that the observer cannot influence. At each moment of discrete time t, the value of some function $F_t(\cdot)$ is observed.

Then the task of tracking changes (tracking) of the optimal parameters of the non-stationary mean risk functional consists in constructing the estimate $\hat{\theta}_t$ of unknown vector θ_t from the sequence of noisy observations $\{y_t\}$ constructed from the points $\{x_t\}$ minimizing the time-varying functional

$$f_t(\theta) = E_{\mathscr{F}_{t-1}} F_t(\theta, w) \to \min_\theta,$$

where $\mathscr{F}_{t-1}$ is the σ-algebra of all probability events that have occurred up to the time $t = 1, 2, ...$, $E_{\mathscr{F}_{t-1}}$ — conditional expectation in the σ-algebra $\mathscr{F}_{t-1}$.

The definition of the SPSA algorithms does not distinguish between its use in stationary and non-stationary cases, but for the non-stationary case, important theoretical results have been obtained for the variant with two observations. Let's introduce the following assumptions for the medium risk functional:

Assumption 1: The difference between successive noise values $\bar{v}_n = v_{2n} - v_{2n-1}, n = 1, 2, ...$, is bounded: $|\bar{v}_n| \leq c_v < \infty$.

Assumption 2: The shift of the optimum point is bounded: $\|\theta_t - \theta_{t-1}\| \leq \delta_\theta < \infty$.

Assumption 3: The rate of displacement of the optimum point is limited in such a way that for any point x : $\|E_{\mathscr{F}_{2n-2}} \nabla \varphi_n(x)\| \leq a_1\|x - \theta_{2n-2}\| + a_0, E_{\mathscr{F}_{2n-2}} \varphi_n(x)^2 \leq a_2\|x - \theta_{2n-2}\|^2 + a_3, \varphi_n(x) = F_{\xi_{2n}}(x, w_{\xi_{2n}}) - F_{\xi_{2n-1}}(x, w_{\xi_{2n-1}})$.

Assumption 4: The functions $f_t(\cdot)$ have unique minimum points θ_t, and for them

$$\forall x \in \mathbb{R}^d : \langle x - \theta_t, E_{\mathscr{F}_{t-1}} \nabla f_t(x) \rangle \geq \mu \|x - \theta_t\|^2,$$

where $\mu > 0$ is a constant, $\langle \cdot, \cdot \rangle$ is the scalar product of vectors.

Assumption 5: The ∇f_t gradient is limited in the root mean squared sense at the minimum points θ_t:

$$E\|\nabla F_t(\theta_t, w_t)\|^2 \leq 0, E\langle \nabla F_t, \nabla F_{t-1}(\theta_{t-1}, w_{t-1}) \rangle \leq 0.$$

Assumption 6: $\forall \xi \in \Xi$ gradient $\nabla f_\xi(x)$ satisfies the Lipschitz condition:

$$\forall x' x'' \in \mathbb{R}^d : \|\nabla f_\xi(x', w_\xi) - \nabla f_\xi(x'', w_\xi)\| \leq M\|x' - x''\|,$$

where M — constant, $M \geq \mu$.

The next two assumptions concern the vector Δ_n and its differentiating kernel $\mathcal{K}_n(\Delta_n)$ in the SPSA algorithm:

Assumption 7: Для любых $n = 1, 2, ...,$

1. Δ_n does not depend on the σ-algebra $\mathscr{F}_{2n-2}$.

2. Noise $\hat{v}_n$ and vector Δ_n are independent of each other.

Assumption 8: For any $n = 1, 2, ...,$ vectors Δ_n and $\mathcal{K}_n(\Delta_n)$ are bounded: $\|\Delta_n\| \leq c_\Delta < \infty, \|\mathcal{K}_n(\Delta_n)\| \leq \kappa < \infty$, and the vector functions $\mathcal{K}_n(\cdot)$, along with the symmetric distribution functions of simultaneous perturbations $P_n(\cdot)$, satisfy the following conditions:

$$\int \mathcal{K}_n(x) P_n(dx) = 0, \int \langle \mathcal{K}_n(x), x \rangle P_n(dx) = I,$$

где I — identity matrix.

A special case that satisfies Assumptions 7 and 8 is the Bernoulli sequence $\{\Delta_n\}$ described earlier and $\mathcal{K}_n(x) \equiv x$.

Let's assume that a sequence of estimates $\hat{\theta}_{2n}$ has an asymptotically effective upper bound $\bar{L} > 0$ of the residuals of an estimate if $\forall \epsilon > 0 \exists N$ is such that $\forall n > N$

$$\sqrt{E\|\hat{\theta}_{2n} - \theta_{2n}\|^2} \le \bar{L} + \epsilon.$$

Without loss of generality, we will interpret the definition of the SPSA algorithm with two observations in such a way that, at each iteration, observations are made at the points

$$x_{2n} = \hat{\theta}_{2n-2} + \beta_n^+ \Delta_n, x_{2n-1} = \hat{\theta}_{2n-2} + \beta_n^- \Delta_n,$$

where $\{\beta_n^+\}$ и $\{\beta_n^-\}$ — sequences of non-negative integers, and $\beta_n = \beta_n^+ + \beta_n^- > 0$.

O.N. Granichin and N.O. Amelins proved in [43] that if assumptions 1–8 hold, $\beta_{max} + \bar{\beta} < \infty$, and the constant α is small enough:

$$k\alpha < 1, \alpha < \frac{\mu}{3\kappa^2(a_2\bar{\beta}^2 + c_\Delta^2 M^2)},$$

then the sequence of estimates produced by the SPSA algorithm with two dimensions has an asymptotically efficient upper bound

$$\bar{L} = h + \sqrt{h^2 + l/k},$$

где $\beta_{max} = \max_n \beta_n, \bar{\beta} = \max_n \frac{1}{\beta_n}, \bar{\beta}^+ = \max_n \frac{\beta_n^+}{\beta_n}, \bar{\beta}^- = \max_n \frac{\beta_n^-}{\beta_n}, k = 2\mu - 6\alpha\kappa^2(a_2\bar{\beta}^2 + c_\Delta^2 M^2), h = \frac{2\delta_\theta}{\alpha k} - \delta_\theta + \frac{\bar{\beta}^- a_1 + 6\alpha c_\Delta^2 M^2(c_\Delta + \bar{\beta}^- \sigma_\theta)}{k}, c_1 = M((\bar{\beta}^+)^2 +$

$3(\bar{\beta}^-)^2), c_2 = M^2(c_\Delta^2 + (\bar{\beta}^+)^2\sigma_\theta^2 + 2\beta^+\beta_{max}\sigma_\theta(c_\Delta + \sigma_\theta)), \bar{l} = 2(\beta^- a_0 + 3\kappa^2\alpha a_3\bar{\beta}^2 + \kappa c_\Delta^2(\beta_{max}c_1 + \kappa\alpha(3c_2 + c_v^2)))$.

Assumptions 2–6 impose restrictions on the non-stationary mean risk functional, but assumption 1 is an assumption on the noise model. Since in the task of optimizing processor energy consumption, the load on the core or cluster affected by computational processes, interpreted as noise, changes within limited limits (from 0 to 100%), if the designed medium risk functional satisfies assumptions 2–6, then due to the above theorem the estimates made by the SPSA algorithm with two noisy measurements will converge to the current optimum point.

2.3 Target function for the simultaneously perturbed stochastic approximation algorithm with single observation in the processor energy consumption optimization problem

The medium risk functional for the SPSA algorithm with a single observation was developed first and a number of assumptions were made in its development. The first is that CPU resources are more often underutilized than 100% utilized. Since, ceteris paribus, the frequency of a CPU determines the amount of work it can do in a given amount of time, then by decreasing the operating frequency, we increase the amount of time it takes to complete the same computational task.

Let the load L on the CPU core or CPU cluster be defined as a percentage of active time of the total CPU time (active and inactive) for some period of time. Similar to the approach in the OnDemand governor, the load on the CPU cluster is defined as the majorant of the loads of the cores included in the cluster. The general idea to be implemented using the SPSA algorithm is to set the frequency for the next iteration of the DVFS loop based on the current state of the processor in terms of load level L so that the load level observed during the next iteration of the DVFS loop is as close as possible to the preselected L_T value. This value should not be close to 100%, so that additional computing resources can be allocated if the actual load turned out to be higher than predicted, and should not be close to 0%, since then even a small load will be calculated at an energy inefficient high frequency.

The second assumption is due to the specifics of the CPU operation. Since both homogeneous and heterogeneous CPUs operate only at a fixed $Freq$ frequency set for each processor or core cluster, the continuous estimate of the optimal frequency is rounded up to the nearest available value.

The objective function is defined as

$$F(f) = 2^{((\text{workload}(f) - L_T)/2)} + \gamma 1.5^{\text{table}(f)},$$

where workload(f) — CPU or CPU cluster load obtained from OS metrics, L_T — target load level, table(f) — frequency number in an array of frequencies available to the CPU or CPU cluster, sorted in ascending order. The first term of the function is the penalty for too low load compared to L_T. Since the same computational load will take less time with increasing frequency, the workload(f) function is non-increasing. The second term is the penalty for using too high a frequency from the available set, and it is monotonically increasing [13].

To set a new operating frequency, calculate

$$f_n = \mathscr{P}(\hat{f}_{n-1} + \beta\Delta_n),$$

where $\mathscr{P}$ — projection to $Freq$, $\hat{f}_{n-1}$ — current frequency estimate, β — SPSA step parameter. For a new frequency estimate we calculate

$$\hat{f}_n = \mathscr{L}(\hat{f}_{n-1} - \frac{\alpha}{\beta}\Delta_n y_n),$$

where $\mathscr{L}$ — projection into the interval $[min(Freq), max(Freq)]$, α, β — step parameters of the SPSA algorithm [12]. This algorithm uses constant values of both parameters from iteration to iteration.

Within the framework of the proposed approach, the change in the computational load caused by the start and end of applications is interpreted as a change in the optimum point in the non-stationary medium risk functional. On the other hand, the 10 ms DVFS cycle time is longer than many computational processes running in modern operating systems and on mobile devices. System interrupts and user event handlers in applications usually last no more than a few milliseconds, but it is often impossible to predict their occurrence at any given time from inside the operating system, for example, it is impossible to predict when the user touches the screen. The change in the computational load caused by such short-term factors is interpreted within the framework of this approach as an additive noise of measurements v_n.

As a result, the DVFS cycle for the SPSA1 governor looks like this:

1. Set initial estimate value $\hat{f}_0$, select α and β.
2. Generate Δ_n.
3. Add a perturbation to the current estimate $f_n = \mathscr{P}(\hat{f}_{n-1} + \beta\Delta_n)$.
4. Set current frequency to f_n.
5. Get new noisy observation $y_n = F(\hat{f}_{n-1} + \beta\Delta_n) + v_n$
6. Update frequency estimate $\hat{f}_n = \mathscr{L}(\hat{f}_{n-1} - \frac{\alpha}{\beta}\Delta_n y_n)$.
7. Go to step 2.

To show validity of algorithm estimates, let's show that the target function satisfies the requirements stated in section 2.1.2. First, due to the specifics of the subject area, changes in the optimal frequency are limited:

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

Further, $F(f)$ satisfies both assumptions that are necessary to prove the consistency of algorithm estimates:

Assumption 1: The function $F(f)$ is strictly convex and has a minimum point f^*:

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

Assumption 2: The gradient $\nabla F(f)$ satisfies the Lipschitz condition:

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

2.4 Target function for the simultaneously perturbed stochastic approximation algorithm with two observations in the processor energy consumption optimization problem

2.4.1 Cost of program execution and medium risk functional

From a theoretical point of view, when solving the same problem, different definitions of the medium risk functionals can differ significantly, but lead to the same solution [44]. In determining the medium risk functional for SPSA2, the results obtained in the study of SPSA1 were used, and they were further developed.

As in the case of SPSA1, we define the load L of the CPU core or CPU cluster as the percentage of active time of the total CPU time (active and inactive) over a certain period of time. In a situation where the frequency is set simultaneously for all cores of the cluster, we will define not the load of an individual core, but the load of the cluster as the majorant of the loads of all the cores included in it. Let's define *computational volume* or simply *volume* as the product of frequency and load. In essence, volume is the number of CPU cycles allocated to an active computing process from its execution to its removal from execution. Although volume correlates with the number of instructions executed in a process, the correspondence is not one-to-one: pipelining allows the processor to execute different phases of sequential instructions at the same time, and the duration of different instructions, measured in CPU cycles, can be different. For the DVFS process, it is important that the same volume creates a different load at different frequencies: the higher the frequency, the lower the load, and vice versa.

Determining the frequency in the DVFS loop for the next iteration of the loop is non-trivial in the sense that we can base our strategy on, among other things, the history of observing CPU load and on the characteristics of OS processes, as is done in EAS, however, the load in the future cannot be reliably determined — for example, it is impossible to predict that the device will receive a phone call in the next second. In view of this uncertainty, we introduce *target* or *threshold load* L_T — a specific constant value of the load during some observation time. Empirically determined values of L_T from 60% to 80% are found in the literature, and they were also used in our experiments — such target levels guarantee additional computing resources in a scenario where the actual load turned out to be much higher than expected from history observations. L_T is used as a boundary between two different DVFS strategies:

- If the current load exceeds L_T, then the core is performing a long enough computational task. In this case, performance considerations outweigh the energy savings. Therefore, the allocation of resources to solve this problem is a priority, and the optimal frequency that the DVFS governor should set is the one that provides the closest load to L_T at the current observed volume. In the long term, if the load does not fall below L_T, higher and higher operating frequency will be set up to the maximum, and this strategy will be used until the load drops.

- If the load does not exceed L_T, this means that if this trend continues, it is possible to optimize power consumption by finding a frequency that provides optimal energy efficiency, and the predicted load still does not exceed L_T.

Let's define *execution efficiency* or *execution cost* (cost of execution, CoE) of the operating frequency f as follows:

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

where CoE is cost of execution, t is the duration of the observation time period, $E(t)$ is the energy expended over the time period t, n_{ticks} is the number of ticks CPU observed during this time. In fact, CoE is the energy cost per cycle of the processor core at a certain operating frequency.

CoE can also be thought of as the priority of the DVFS governor when selecting a frequency — the lower the value, the higher the priority. From this point of view, instead of CoE, one can use some *generalized cost of execution* ($GCoE$) metric to determine the priority when choosing frequencies. In this case, $GCoE$ is defined as a mapping of a finite set of CPU cluster frequencies into some set for which a linear order relation is defined. The strictness of this relationship is optional: if several operating frequencies have the same priority, then the choice of one of them remains at the discretion of the DVFS governor [78].

In this study, $GCoE$ is defined as the ratio of the average current when the core is running at a certain frequency to that frequency:

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

As the operating voltage of the processor can be considered constant, so removing the constant factor U from the definition of CoE does not change the order ratio when going to $GCoE$[3].

The key point that influenced the study and use of CoE and $GCoE$ in this work is the fact that for the available power consumption profiles of CPU clusters, as f grows, so does I_f, but $GCoE(f)$ in the general case is a non-monotonically increasing function. An example of such behavior is shown in Tab. 1.1, where the A55 core consumes the least amount of power at 500 MHz, but is most energy efficient at 875 MHz.

Thus, when the current load falls below L_T, the optimal frequency will be the one with the lowest $GCoE$ among the frequencies capable of handling the same amount of predicted load without exceeding L_T.

Combining both operating modes of the DVFS governor (performance mode and power saving mode) together, we define the result of calculating the medium risk functional and the optimality of the current operating frequency as the distance between the indices of the current and optimal frequencies in the sorted list of frequencies:

$$F(f) = \begin{cases} \|\mathscr{P}(f) - \mathscr{P}(f\frac{L}{L_T})\|, L_T < L \leq 1 \\ \|\mathscr{P}(f) - \mathscr{S}(\min_{f_1..f_n} GCoE(f_i) : L\frac{f}{f_i} <= L_T)\|, 0 \leq L \leq L_T, \end{cases}$$

where $\mathscr{P}(\cdot)$ is the projection to the index of the nearest frequency in the array of frequencies, sorted in ascending order, $\mathscr{S}(\cdot)$ is the projection of frequency $GCoE$ to its index in an array of frequencies sorted in ascending order.

The load level L can vary from measurement to measurement, however,

[3]Moreover, for used devices U=1V, so CoE and $GCoE$ were numerically equal.

for such a definition of the target function, it is necessary to consider a special case when one noisy measurement corresponds to the power saving mode, and the other to the performance mode. This situation occurs when a period of intensive work ends, or, conversely, a new computationally intensive process begins. To overcome this obstacle, we will assume that in such a situation internally consistent conclusion cannot be made about the upcoming load, therefore we will discard this pair of measurements without further changing the frequency in this iteration of the DVFS cycle, and from the next iteration we will start a new iteration of the SPSA algorithm according to the chosen scheme.

Note that the device lack of frequencies with the same $GCoE$ means that for any observed L, the optimal frequency will always be unique. From a practical point of view, if such a situation occurs, then the collision can be resolved by assigning a smaller frequency to a smaller $GCoE$.

Although the target function knows which frequency is best to switch to at the moment “inside itself”, this information is deliberately not taken “outside” so that the algorithm takes a step towards the optimal frequency, but does not immediately set it. Such a strategy leads to better adaptability of the algorithm both in the situation when the prediction of the algorithm coincided with reality, and in the situation when the prediction was not justified.

2.4.2 Obtaining noisy observations

Based on the definition of SPSA2, its implementation requires two noisy observations, after which the current optimal estimate is established, which results, for the purposes of this study, in three iterations of the DVFS loop:

1. Let i_i be the index of the current operating frequency in an array of frequencies sorted in ascending order. Let’s generate a random number $\Delta = \pm 1$, after which we set the current frequency with index $i_i + \Delta\beta$ to obtain the first noisy observation y^+ of the medium risk functional.

2. Next, the frequency with index $i_i - \Delta\beta$ is set as the current frequency to obtain the second noisy observation y^-.
3. The new optimal frequency estimate $i_{i+1} = i_i - \frac{\alpha(y^+ - y^-)}{2\Delta\beta}$ is set as current.

Note that due to the lack of floating-point operations in the Android OS kernel, it is easier to work with frequency indices in a sorted array than with the frequencies themselves. In all cases, when the index goes beyond the bounds of the array, the lower or upper bound, respectively, is taken as the result of a subtraction or addition operation.

However, in the very definition of SPSA2 there is an opportunity for optimization — setting a new estimate of the optimal frequency can be interpreted as the first noisy observation on the next cycle, and therefore the cycle of the algorithm will require only two iterations of the DVFS cycle:

1. The current value of the function at the selected frequency with index i_i is used as the first noisy observation y^0. A random number $\Delta = \pm 1$ is generated, and to obtain the second noisy observation of the function y^+, the frequency with index $i_i + \Delta\beta$ is set.
2. A new frequency is set with index $i_{i+1} = i_i - \frac{\alpha(y^+ - y^0)}{\Delta\beta}$.

However, waiting for two or three DVFS cycles to complete one iteration of the algorithm can be detrimental to the accuracy of the estimate in dynamically changing environment and, as a consequence, to the quality of the governor. Therefore, a variant of the algorithm is proposed that takes only one iteration of the DVFS loop, although it deviates from the strict definition of SPSA2:

1. Let i_i be the index of the current operating frequency in an array of frequencies sorted in ascending order. Let's generate a random number $\Delta = \pm 1$. The load estimate for frequencies with indices $i_i \pm \Delta\beta$ is

calculated under the assumption that the volume remains unchanged, and for the simulated estimates, the function values y^+ and y^- are calculated. Next, the operating frequency is set with the index $i_{i+1} = i_i - \frac{\alpha(y^+ - y^-)}{2\Delta\beta}$.

Moving forward, each scheme will be called, based on the number of DVFS cycles required for one cycle of the algorithm, SPSA2_3, SPSA2_2 and SPSA2_1, respectively.

Chapter 3. Processor Frequency Governors Based on the Simultaneously Perturbed Stochastic Approximation Algorithms with One and Two Observations

This chapter describes the technical peculiarities of the development of DVFS governors based on the SPSA algorithm and the medium risk functionals from Chapter 2, outlines the methodology and results of comparing the developed governors with existing ones, and draws conclusions from the results of their testing.

3.1 Details of governors based on the simultaneously perturbed stochastic approximation algorithms with one and two observations programming

The SPSA1 governor and all three schemes for the SPSA2 governor were implemented on the basis of the infrastructure of the standard OnDemand governor in the C programming language. During development, it was necessary to take into account that the Android kernel, like the Linux kernel on which it is based, does not support floating-point operations, therefore, if necessary, functions from a floating-point argument were translated to use integer arguments. Because of this, in a number of cases, tabular definition of functions, including continuous functions, was used if it was known that, within the limits of the algorithm, they could be calculated from a finite number of different arguments. Additional checks for going beyond the array of frequencies boundaries were made in all cases before setting the current estimate of the optimal frequency.

It should be especially noted that within the framework of the dissertation research, in all developed governors, the energy consumption constants or $GCoE$ metrics for all frequencies of all clusters of specific CPUs of test benches were written as constants in the program code. A more correct approach from an engineering point of view would be to modify the OS kernel, which would provide these constants by analogy with information on supported frequencies, however, due to noticeable differences in the code of Android OS kernels from different manufacturers, it is advisable that such a programming interface be initially provided by the kernel development team.

For the SPSA2_1 scheme, an empirical optimization for performance mode has been added. If the load within the last observation period is 100%, the value of β is changed so that the operating frequency increases more significantly. It does not contradict the idea of SPSA2, since in the general case, at each iteration, the values of the β_n sequence can be used, rather than a specific constant β. If the high load level did not last long, and the decision to change β turns out to be excessive in terms of power consumption, then the operating frequency also decreases rapidly, and β returns to its original value.

The time required for the processor to change the frequency may exceed the duration of the DVFS loop iteration. For example, the default OnDemand governor DVFS cycle time is 10 ms. But since OnDemand was designed for desktop computers and servers with more advanced CPU circuitry, this interval was enough to switch the operating frequency. At the same time, during preliminary experiments with Xiaomi Redmi Note 8 Pro, it was found that after the initial request to change the frequency with the OnDemand governor or another DVFS governor based on it, it may take up to three consecutive DVFS calls for the updated CPU frequency to be displayed in the OS APIs. In this case, the DVFS governor can send a new frequency change request with each call, which will be ignored until the initial one is executed. The observed phenomenon suggests that even the most common DVFS governors can be based on assumptions and invariants that may be false for the devices they control.

Since the developed SPSA2 governors also use OnDemand clock model, and it is critical for the algorithm to obtain noisy observations at target frequencies, the program skips those DVFS calls for which the CPU cluster frequency still does not match the requested one. The algorithm continues if and only if the requested frequency is set. Additionally, it is assumed that within the last DVFS cycle, at the end of which the requested frequency was reflected in the OS, the requested frequency was active for the entire duration of its duration, and due to this, no changes are made to the process of estimating the noisy load of the CPU cluster. For the SPSA1 governor and the SPSA2_1 scheme, this factor is not so relevant due to the coincidence of the duration of the algorithm and the DVFS cycle, and therefore was not taken into account. However, taking into account this phenomenon can be used to slightly reduce the running time of the algorithm.

The EAS task scheduler, which takes into account energy efficiency in its work, actively interacts with the DVFS schedutil governor. According to the documentation, EAS is not guaranteed to reduce power consumption unless schedutil is selected as the active governor [55]. From an algorithmic point of view, schedutil implements the same strategy as the OnDemand governor, that is, it selects the cluster frequency in proportion to the load majorant on the cluster cores. The minimum frequency corresponds to a load of 0%, and the maximum frequency corresponds to a load of 100%. However, the definition of CPU load in EAS differs significantly from that used in OnDemand. CPU cluster load is estimated in terms of the EAS power consumption model and the compute capacity of all tasks assigned to a particular CPU core. Instead of a posteriori estimate, it is estimated a priori.

For the proposed schemes, load detection was used as in OnDemand even with EAS enabled, and, as experiments have shown, the selection of α, β and L_T parameters is sufficient for SPSA2 to work effectively with EAS.

For the software implementation of three SPSA2 schemes, a certificate of state registration of a computer program was received with № 2023666564 named “DVFS energy-saving governors for Android OS based on the SPSA algorithm with two dimensions and the OnDemand governor” from August 2nd, 2023 [91] (see Appendix B).

3.2 Results of testing the developed governors based on the simultaneously perturbed stochastic approximation algorithms with one and two observations

3.2.1 Results of testing the governor based on the single observation algorithm

Xiaomi Redmi Note 8 Pro smartphone was chosen for testing SPSA1. It has Helio G90T heterogeneous CPU with two core clusters — 2 Cortex-A76 (big) cores and 6 Cortex-A55 (LITTLE) cores. The default OS is Android 10. This model was chosen due to the existing set of utilities for running alternative builds of the Android OS kernel on the device. The begonia[4] kernel was taken as the basis for the experimental build, and the SPSA1 governor was added to the list of DVFS governors. Root access was required to install the experimental build. Switching to the SPSA1 governor was performed by standard system calls of the `cpufreq` subsystem [108].

OnDemand and Interactive were selected as governors for comparison, CFS as the task scheduler, and menu as the idle state governor.

[4] `https://github.com/AgentFabulous/begonia`

Since the goal of the work is to create a general purpose DVFS governor adapted to handle various loads, a set of test cases reflects various patterns of human-computer interaction:

1. Playing *Hill Climb Racing*[5].
2. Playing song in *Spotify*.
3. Displaying and periodically scrolling through a PDF file opened in *Foxit PDF Editor*[6].
4. Watching video on *YouTube* with a quality setting of 480p in a default browser.
5. Watching video on *YouTube* with a quality setting of 1080p in a default browser.

To run and control test scenarios, the device was connected via USB to the host computer.

The testing methodology was prepared according to the recommendations collected in [73]:

- All applications not required for operation or testing purposes have been removed. Apps that can't be uninstalled have their background activity disabled or reduced as much as possible.
- All smartphone components not used in the current test, such as Wi-Fi, 3G, GPS, were turned off.
- Before starting each subsequent test, a pause of two minutes was taken to allow background processes on the device, possibly caused by the launch of the test, to end, and the device itself to cool down.

[5] `https://play.google.com/store/apps/details?id=com.fingersoft.hillclimb`

[6] `https://play.google.com/store/apps/details?id=com.foxit.mobile.pdf.lite`

Table 3.1: Comparison of energy consumption of SPSA1, OnDemand and Interactive governors

Test case	SPSA1 (in mAh)	OnDemand (in mAh)	Interactive (in mAh)
Hill Climb Racing	**34.196**	91.862	37.565
Spotify	33.913	**24.678**	41.343
Foxit PDF Editor	**54.729**	65.322	77.950
YouTube 480p	32.957	**24.579**	47.980
YouTube 1080p	32.378	**25.319**	47.254

- Each script was run five times for fifteen minutes to reduce the impact of background processes.
- Before each series of runs of each test scenario, the scenario was run once, and the power consumption during its passage was not taken into account. Thus, the test scenario was warmed up in order to exclude one-time processes from further launches that could affect power consumption (for example, loading software libraries).
- Since the device was connected to USB, it was operating in external power mode, so the battery was not recharged to the same initial level.

The Google base model was used as the energy estimation model. To compare the performance, we used the `AnTuTu`[7] benchmark — a utility for evaluating device performance that is widespread among Android OS developers and provides integral evaluation of device performance under various computing loads for the CPU, GPU, memory and human-machine interface responsiveness (User Experience, UX).

Tab. 3.1 contains averaged data on energy consumption based on the results of experiments, converted to mAh, best results are highlighted in **bold**.

In the *Hill Climb Racing* script, OnDemand performed the worst — it always brings the A55 and A76 cores to the maximum frequency. SPSA1

[7]`https://www.antutu.com/en/index.htm`

Table 3.2: AnTuTu performance scores for Powersame, SPSA1, OnDemand, Interactive, and Performance governors

Governor	CPU	GPU	Memory	UX	Total
SPSA	80555	74357	40897	48442	244251
OnDemand	**83704**	76335	**42020**	**49893**	**251972**
Interactive	79459	**76880**	39403	49412	245154
Performance	86756	79952	42364	51789	260861
Powersave	28883	50193	30792	18458	128326

performs slightly better than Interactive because it keeps the A55 at lower frequencies, although the A76 has a wider used frequency range.

In the *Spotify* scenario, Interactive considers that the need for performance on the A76 cores is greater than necessary and overestimates the frequency to set, SPSA1 uses all available frequencies for A55, while OnDemand keeps the operating frequency closer to the lower boundary.

Although the *Foxit PDF Editor* script does not appear to be the most CPU-intensive at first glance, rendering a new page in a PDF file is computationally expensive. Here SPSA1 wins, since the A76 cores are not actively used in work.

The difference in power consumption between the *Youtube 480p* and *YouTube 1080p* scripts is negligible, even though significantly more data is transferred during the second test. Interactive sets the A76 cores to the maximum frequency and therefore loses to other regulators in terms of power consumption. OnDemand is more conservative in its estimates for both A55 and A76 cores than SPSA1, so it consumes less power.

In Fig. 3.1 and 3.2 the difference in the behavior of the governors in the Spotify test is depicted in more detail in the form of distribution histograms of time spent by clusters at a certain frequency[8]. It can be seen that fundamentally different strategies for setting the operating frequency of the governors under consideration lead to fundamentally different time distribution profiles and, as a result, energy consumption.

[8]Histograms for all experiments are available at `https://drive.google.com/file/d/`

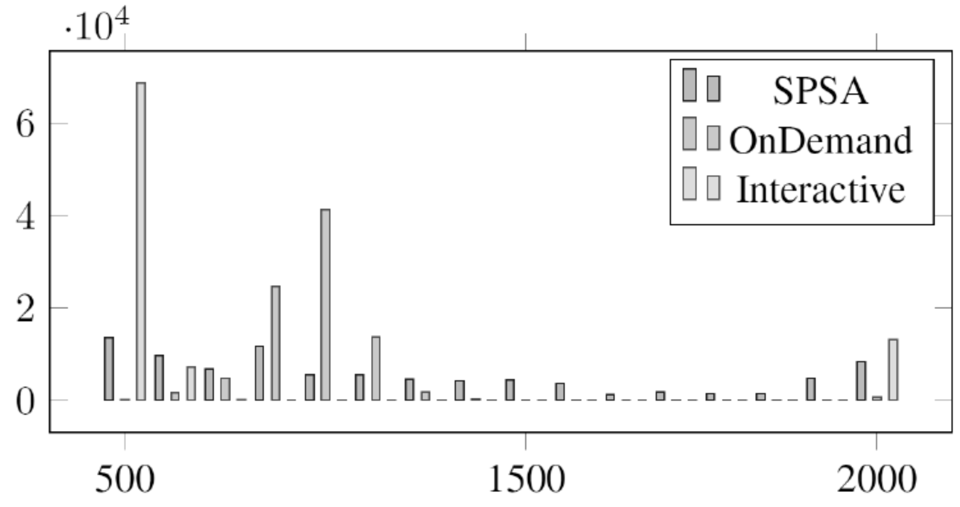


Figure 3.1: Distribution of times (in ms) spent by the A55 cluster running various DVFS governors at various frequencies (in MHz) in the Spotify test

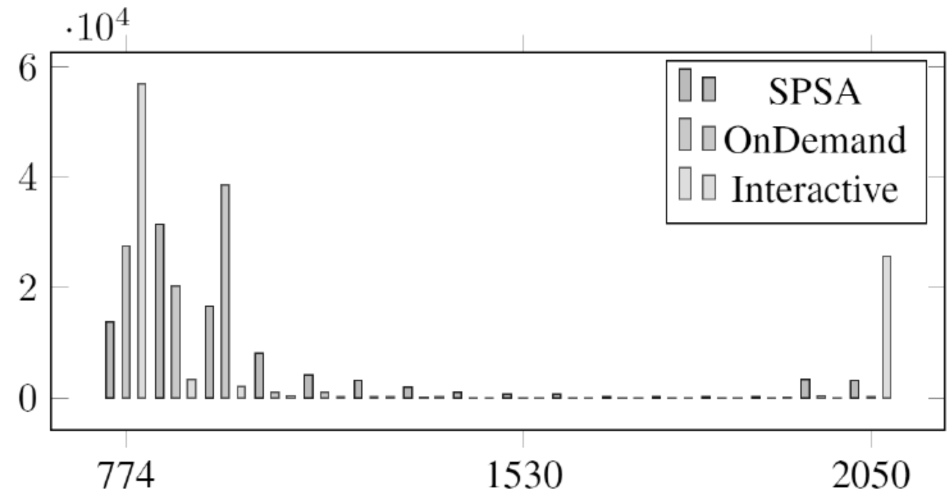


Figure 3.2: Distribution of times (in ms) spent by the A76 cluster running various DVFS governors at various frequencies (in MHz) in the Spotify test

Tab. 3.2 shows the performance scores of SPSA1, OnDemand and Interactive by the `AnTuTu` utility in the four listed categories, and the "Total" column shows their sum as an integral quality metric, best results are highlighted in **bold**. The results of the `Powersave` and `Performance` controls are shown as the minimum and maximum achievable values.

The performance difference between SPSA1, OnDemand and Interactive

17ki2HxrFVwYjccV4-pPwIZuD8g8iqoLG/view

is almost imperceptible to the end user (3.06% and 0.36% compared to OnDemand and Interactive respectively), and the performance achieved by SPSA1 is comparable to the best achievable (6.38% difference) with `Performance` governor). When comparing the performance metrics of GPU, memory and human-machine interface responsiveness, it can be seen that the strategy of the DVFS regulator indirectly affects the performance of other components of the mobile device and, consequently, the performance of the entire system as a whole.

3.2.2 Results of testing the governor based on the two observations algorithm

The Samsung Galaxy s7 SM-G930F smartphone was used to test SPSA2. The Exynos 8 Octa (8890) processor installed on it is built on a heterogeneous architecture with two clusters: 4 Cortex-A53 (LITTLE) cores and 4 Exynos M1 (big) cores.

Although it originally ran Android 8, Android 10 and 11 OS builds were installed for testing. Android 11 there uses the EAS task scheduler. For testing with the CFS scheduler, the kernel build from Samsung Corporation — android_kernel_samsung_universal8890 for Android 10[9] — was used. Thanks to this choice of kernel builds, it became possible to add the schedutil governor to the comparison with the previously used Interactive and OnDemand governors.

For the most part, the preparation of the test bench and the methodology of the experiments remained unchanged. As before, the $SPSA2_3$, $SPSA2_2$, and $SPSA2_1$ governors were added as separate modules to the assembly, and switching between governors was also done using standard `cpufreq` subsystem system calls.

The choice of test scenarios for this series of experiments was dictated by similar considerations:

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

- Playing an MP4 file using the *VLC video player* that was previously selected as the default player.
- Game session in *Trial Xtreme 3*.
- Game session in *Flapy Bird*.
- Create a note in the *Notes* application, write text, and delete the note.
- Launch the default *camera* application and record video with it. The resulting video file is deleted at the end of the script.
- View a video stream in the *Twitch* service in a browser and close the browser after some time.

As before, to run and control test scenarios, the device was connected via USB to the host computer and therefore was powered by an external power source.

The preparation of the device for running the test scenarios followed the same principles as in Section 3.2. The only difference was that the tests were run ten times for five minutes per run — experience with SPSA1 showed that at a five-minute interval the effect of background processes is already negligible.

Unlike the previous series of experiments, SPSA2 was tested using both the base power consumption model from Google and the one proposed in this study. In addition, the Geekbench 5.5.1 utility was used to test the performance of the device, since its methodology, unlike `AnTuTu`, is open. This methodology was suitable for the research [29]. Due to the duration, performance tests were run 3 times for each governor, and the average values of the metrics were found.

First, experiments were carried out for Android 11 with the EAS task scheduler. The selection of integer parameters of the SPSA algorithm showed that the following sets are the best:

Table 3.3: Energy consumption (mAh) under Android 11 and EAS scheduler (base energy model), videoVLC, trialXTreme3 and flappyBird test cases

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 3.4: Energy consumption (mAh) under Android 11 and EAS scheduler (base energy model), notes, camera and twitch test cases

Algorithm	notes	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

Table 3.5: Energy consumption (mAh) under Android 11 and EAS scheduler (modified energy model), videoVLC, trialXTreme3 and flappyBird test cases

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 3.6: Energy consumption (mAh) under Android 11 and EAS scheduler (modified energy model), notes, camera and twitch test cases

Algorithm	notes	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

Table 3.7: GeekBench 5.5.1 performance scores

Algorithm	Single core mode	Multicore mode
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

1. $\alpha = 2, \beta = 1, L_T = 70\%$;
2. LITTLE cluster: $\alpha = 2, \beta = 1, L_T = 80\%$; big cluster: $\alpha = 3, \beta = 1, L_T = 98\%$.

Tab. 3.3–3.6 contain median energy consumption values for the base and modified energy consumption models. Tab. 3.7 contains performance data. Best results are highlighted in **bold**. The first line for SPSA2 governors corresponds to the first set of parameters, and the second line corresponds to the second set.

It should be noted that the obtained values for the SPSA and schedutil regulators have a noticeably lower spread in all experiments compared to OnDemand and especially Interactive. This is due to the fact that energy consumption is compared by the median and not by the average value. For example, Interactive showed the minimum and maximum values in the videoVLC test with a base model of power consumption of 42.42 and 155.91 mA·h, respectively. However, the observed phenomenon does not affect the reliability of the analysis, since even the minimum power consumption of both regulators was higher than the maximum value for schedutil or SPSA.

In terms of performance, both existing and new algorithms perform comparably. There are no obvious outliers, and for a single-core scenario, the difference between the best and worst performance is within 15%, and for a multi-core scenario, within 18%. Note that the first set of SPSA2 parameters provides better performance than the second set. The performance of SPSA2_3 is worse due to the fact that the algorithm requires three iterations of the DVFS loop, accompanied by waiting for a frequency switch.

Before considering energy consumption estimates, it should be noted that the modified energy consumption model significantly changes the observed values compared to the base model, and in some cases their relative standing between individual test cases. The modified model, in our opinion, better reflects reality if direct measurement of energy consumption is not available, and further analysis is done on its basis, despite the fact that SPSA2 governors are evaluated to be better under the base energy model.

Among the standard governors, Interactive and OnDemand consistently consume more power than schedutil when using EAS, and if EAS is present in the device firmware, it is recommended to use the EAS+schedutil pair to save power consumption and at the same time keep the human-machine interface responsive.

SPSA2 governors handle *Trial Xtreme 3* and *Twitch* scenarios better than schedutil. This is due to the use of the $\beta = 1$ parameter in the experiments, since higher values of β lead to faster gradient descent, and in a situation of rapidly changing processor load with frequency switching delays, a relatively smooth change in the operating frequency shows the best result. SPSA2_1 with the first set of options consumes more power in most tests (up to 27%), but is only 10% better in performance than schedutil, so it is recommended to use the second set of options, as the performance stays at the same level, and power consumption can be up to 31% better. SPSA2_2 is the most balanced governor, because while its power consumption is up to 20% better in some scenarios and 10% worse in others, a device running it shows a performance gain of 5.6% in the multi-core case, which is a more realistic scenario in terms of modern patterns of human-machine interaction. SPSA2_3 is the most conservative of the other SPSA family governors in terms of power consumption and performance, but overall multi-core performance is comparable to schedutil.

A special case is the videoVLC scenario, where SPSA governors use more power than schedutil, although the difference is not too great (no more than 7.2% for SPSA2_2). The ratio of the duration of intensive calculations (frame decoding) and idle time is such that the SPSA regulators do not have time to lower the frequency. The only exception is SPSA2_3, but the difference is within the statistical error.

In the end, when working with the EAS scheduler, all schemes of the SPSA2 governor can be used instead of schedutil.

Table 3.8: Energy consumption (mAh) under Android 10 and CFS scheduler (base energy model), videoVLC, trialXTreme3 and flappyBird test cases

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 3.9: Energy consumption (mAh) under Android 10 and CFS scheduler (base energy model), notes, camera and twitch test cases

Algorithm	notes	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

Table 3.10: Energy consumption (mAh) under Android 10 and CFS scheduler (modified energy model), videoVLC, trialXTreme3 and flappyBird test cases

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 3.11: Energy consumption (mAh) under Android 10 and CFS scheduler (modified energy model), notes, camera and twitch test cases

Algorithm	notes	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

Tab. 3.8-3.11 contain the median power consumption values for the base and modified energy consumption models for scenarios executed on a device running Android 10 OS and the CFS scheduler. Best results are highlighted in **bold**. The situation in this case is significantly different. First, schedutil cannot be used as a benchmark because it will work in conditions for which it was not designed. The Interactive governor shows a noticeable improvement in power consumption. All SPSA2 governors exhibit similar energy behavior to the SPSA1 governor, where in most test scenarios they are between OnDemand and Interactive in terms of power consumption, and in some individual cases they perform better or worse.

The reason for this energy behavior is a change in the planning strategy. Unlike EAS, CFS does not prioritize LITTLE cores over big cores, and in such situations, big cores are loaded more, which in turn leads to more power consumption. This trend becomes especially noticeable in heterogeneous configurations, where the number of big cores is equal to or comparable to the number of LITTLE cores.

While $SPSA2_3$ shows acceptable power consumption in general, we found that the performance of the device under its control is lower, since at load levels of 99—100% it does not immediately recommend increasing the frequency. Because other algorithms exhibit higher overall power consumption than Interactive, which is consistent with higher clock speeds, and the latter provides acceptable human-machine interface responsiveness, performance tests have not been conducted for SPSA2 governors.

3.3 Analysis of governors test results

The first conclusion that can be drawn from the results of developed governors testing is that, based on all variations of the simultaneously perturbed stochastic approximation algorithms, it is possible to construct a DVFS governors suitable for everyday use in modern mobile devices. Moreover, although the governors were designed for heterogeneous CPU architectures, they can be used in devices with a homogeneous architecture.

The algorithms with which the comparison was made are deterministic and try to find the optimal solution under conditions of uncertainty. At the same time, the analysis of the results shows that governors based on randomized algorithms can, in some cases, make non-optimal decisions based on the available data, which actually turn out to be correct. Thus, obtaining an optimal result on average in the long term turns out to be a competitive advantage of the developed governors.

Experiments show that one adequately selected set of parameters for each cluster is enough to build an energy-saving general-purpose governors that can operate in a wide range of loads. At the same time, the different behavior of both standard and developed governors shows that different computational loads load the CPU in different ways. Earlier it was mentioned that the applied software can have the function of energy profiles, that is, change its energy behavior depending on the battery charge level [86]. By analogy with this approach, the set of DVFS governor parameters that is optimal in terms of power consumption and performance for a particular application can be called an energy profile on the system software level. Taking into account the results obtained in this study, the task of OS evolution for the automatic or user-directed usage of such energy profiles for various loads ceases to be conceptual and becomes purely technical.

Conclusion

The main scientific results of the dissertation obtained as a result of achieving the research tasks:

1. A model for estimating the power consumption of a CPU built on a heterogeneous architecture is proposed and justified, taking into account the dynamic computational load and idle states management, and practical recommendations for its application are given.

2. An approach has been developed for solving the problem of power consumption management for a heterogeneous CPU based on randomized stochastic optimization algorithms, within which average risk functionals for SPSA algorithms with one and two measurements are proposed and justified. Theoretical validity of the estimates provided by the developed algorithms is investigated and established within the limits imposed by the features of the CPU operation.

3. DVFS modules have been developed that optimize CPU power consumption, taking into account the peculiarities of its operation, based on the proposed functionals and SPSA algorithms with one and two measurements. A comparison with existing analogues has been made on the basis of prepared test benchmarks running in the Android OS. For modules based on SPSA algorithms with two observations, a certificate of state registration of computer programs has been obtained.

Bibliography

[1] *Ahmad R. W. et al.* A case and framework for code analysis–based smartphone application energy estimation // International Journal of Communication Systems. – 2017. – Vol.30, №. 10.

[2] *Ahmad R. W. et al.* Enhancement and assessment of a code-analysis-based energy estimation framework // IEEE Systems Journal. — 2018. — Vol. 13, no. 1. – P. 1052–1059.

[3] *Aldahlawi A., Kim Y.B., Kim K.K.* GPU architecture optimization for mobile computing // 2019 International SoC Design Conference (ISOCC). -– IEEE, 2019. -– P. 247–248.

[4] *Amelin K.S. et al.* Adaptive control of an autonomous group of unmanned aerial vehicles // Stochastic optimization in informatics. -– 2009. — Vol. 5. -– P. 157–166.

[5] *Amelina N. et al.* Approximate consensus in stochastic networks with application to load balancing //IEEE Transactions on Information Theory. – 2015. – Vol.61. – №. 4. – P. 1739–1752.

[6] *Amelina N. O., Ivanskiy Yu. V.* The task of achieving a differentiated consensus under cost constraints // Bulletin of St. Petersburg University. Mathematics. Mechanics. Astronomy. — 2015. — Vol. 2. — № 4. — P. 495–506.

[7] *Banerjee A. et al.* Detecting energy bugs and hotspots in mobile apps // In: Proc. of the 22nd ACM SIGSOFT international symposium on foundations of software engineering. – 2014. – P. 588–598.

[8] *Bareth U.* Simulating power consumption of location tracking algorithms to improve energy-efficiency of smartphones // In: Proc. of 2012 IEEE 36th Annual Computer Software and Applications Conference. – IEEE, 2012. – P. 613–622.

[9] *Basireddy K. R. et al.* AdaMD: Adaptive mapping and DVFS for energy-efficient heterogeneous multicores // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2019. – Vol. 39, no. 10. – P. 2206–2217.

[10] *Beard K.W.* Linden's Handbook of Batteries, 5th Edition. – McGraw-Hill Education, 2019. – 1456 p.

[11] *Blum J. R.* Multidimensional stochastic approximation methods // The annals of mathematical statistics. – 1954. – p. 737–744.

[12] *Bogdanov E., Bozhnyuk A., Bykov D., Sartasov S., Sergeenko A., Granichin O.* Dynamic Voltage-Frequency Optimization using Simultaneous Perturbation Stochastic Approximation // In: Proc. of 60th IEEE Conference on Decision and Control (CDC) 2021. – 2021. – P. 3774–3779.

[13] *Bogdanov E., Bozhnyuk A., Sartasov S., Granichin O.* On Application of Simultaneous Perturbation Stochastic Approximation for Dynamic Voltage-Frequency Scaling in Android OS // In: Proc. of 7th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP) 2021. – 2021. – P. 1–7.

[14] Broyde L. et al. MobiCore: An adaptive hybrid approach for power-efficient CPU management on Android devices // In: Proc. of 30th IEEE International System-on-Chip Conference (SOCC 2017). — 2017. – P. 221–226.

[15] *Carette A. et al.* Investigating the energy impact of android smells // In: Proc. of 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). – IEEE, 2017. – P. 115–126.

[16] *Chae S.-H., Yoo Ch.-H, Sun J.-Y., Kang M.-Ch., Ko S.-J.* Subpixel rendering for the pentile display based on the human visual system // IEEE Transactions on Consumer Electronics. 2017. – Vol. 63, № 4. – P. 401–409.

[17] *Chen H.-F., Duncan T. E., Pasik-Duncan B.* A Kiefer-Wolfowitz algorithm with randomized differences // IEEE Transactions on Automatic Control. – 1999. — Vol. 44, no. 3. – P. 442–453.

[18] *Chen X., Zong Z.* Android app energy efficiency: The impact of language, runtime, compiler, and implementation // Proceedings of 2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (socialcom), sustainable computing and communications (sustaincom)(BDCloud-socialcom-sustaincom). – IEEE, 2016. – P. 485–492.

[19] *Chen Y.L., Chang M.F., Yu Ch.W., Chen, X.Zh., Liang W.Y.* Learning-directed dynamic voltage and frequency scaling scheme with adjustable performance for single-core and multi-core embedded and mobile systems // Sensors. – 2018. – Vol. 12, no. 9.

[20] *Chung J. M. et al.* Adaptive Cloud Offloading of Augmented Reality Applications on Smart Devices for Minimum Energy Consumption // Ksii Transactions on Internet & Information Systems. -– 2015. — Vol. 9, no. 8.

[21] *Chung Y. F., Lin C. Y., King C. T.* Aneprof: Energy profiling for android java virtual machine and applications // Proceedings of 2011 IEEE 17th International Conference on Parallel and Distributed Systems. – IEEE, 2011. – P. 372–379.

[22] *Couto M. et al.* GreenDroid: A tool for analysing power consumption in the android ecosystem // In: Proc. of 2015 IEEE 13th International Scientific Conference on Informatics. – IEEE, 2015. – P. 73–78.

[23] CPU frequency and voltage scaling code in the Linux (TM) kernel. Linux CPUFreq. CPUFreq Governors. Available at: `https://android.googlesource.com/kernel/common/+/a7827a2a60218b25f222b54f77ed38f57aebe08b/Documentation/cpu-freq/governors.txt`. (accessed 10.08.2023).

[24] *Cruz L., Abreu R.* Catalog of energy patterns for mobile applications // Empirical Software Engineering. – 2019. – Vol. 24. – P. 2209–2235.

[25] *Dey S. et al.* CPU-GPU-Memory DVFS for Power-Efficient MPSoC in Mobile Cyber Physical Systems // Future Internet. – 2022. – Vol. 14, no. 3.

[26] *Di Nucci D. et al.* Petra: a software-based tool for estimating the energy profile of android applications // Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). – IEEE, 2017. – P. 3–6.

[27] *Dong M., Lan T., Zhong L.* Rethink energy accounting with cooperative game theory // Proceedings of the 20th annual international conference on Mobile computing and networking. – 2014. – P. 531–542.

[28] *Dolezal J., Becvar Z.* Methodology and tool for energy consumption modeling of mobile devices // Proceedings of 2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW). – IEEE, 2014. – P. 34–39.

[29] Geekbench 5 CPU Workloads. Available at: `https://www.geekbench.com/doc/geekbench5-cpu-workloads.pdf`. (accessed 10.05.2023).

[30] *Feghhi M.* Multi-layer tracing of android applications for energy-consumption analysis. – Master Thesis. – 2017.

[31] *Fischer L. M., de Brisolara L. B., De Mattos J. C. B.* Sema: An approach based on internal measurement to evaluate energy efficiency of android applications // 2015 Brazilian Symposium on Computing Systems Engineering (SBESC). – IEEE, 2015. – P. 48–53.

[32] *Gao X. et al.* E-android: A new energy profiling tool for smartphones // 2017 IEEE 37th international conference on distributed computing systems (ICDCS). – IEEE, 2017. – P. 492–502.

[33] *Gasnikov A.V. et al.* Numerical methods for finding the equilibrium distribution of flows in the Backman model and in the stable dynamics model // Mathematical Modeling. – 2016. — Vol. 28. — № 10. — P. 40–64.

[34] *Granichin O. N.* On one stochastic recursive procedure with dependent interference in observation, using trial disturbances at the input // Bulletin of the Leningrad University. Series 1: Mathematics, mechanics, astronomy. – 1989. – № 1. – P. 19–21.

[35] *Granichin O. N.* Estimation of the minimum point of an unknown function observed against the background of dependent noise // Problems of information transmission. – 1992. – Vol. 28. – № 2. – P. 16–20.

[36] *Granichin O. N.* The procedure of stochastic approximation with a perturbation at the input // Automatics and Telemechanics. — 1992. — № 2. — P. 97–104.

[37] *ΓGranichin O. N.* Randomized algorithms for stochastic approximation with arbitrary noise // Automatics and Telemechanics. — 2002. — № 2. — P. 44–55.

[38] *Granichin O. N.* Optimal rate of convergence of randomized stochastic approximation algorithms under arbitrary noise // Automatics and Telemechanics. – 2003. – № 2. – P. 88–99.

[39] *Granichin O. N., Polyak B. T.* Randomized estimation and optimization algorithms for almost arbitrary noise. – M.: Nauka, 2003. – 291 p.

[40] *Granichin O.* Linear regression and filtering under nonstandard assumptions (Arbitrary noise) // IEEE Transactions on Automatic Control. – 2004. – Vol. 49, no. 10. – P. 1830–1837.

[41] *Granichin O. N., Krasnoshchekov V. E.* Energy consumption optimization algorithms in mobile devices // Neurocomputers: development, application. – 2007. – № 6. – P. 81–86.

[42] *Granichin O., Gurevich L., Vakhitov A.* Discrete-time minimum tracking based on stochastic approximation algorithm with randomized differences // Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference. – IEEE, 2009. – P. 5763–5767.

[43] *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. – P. 1653–1658.

[44] *Granichin O., Volkovich Z., Toledano-Kitai D.* Randomized Algorithms in Automatic Control and Data Mining. – 2015. – 249 p.

[45] *Greenhalgh, P.* big.LITTLE technology: The future of mobile [White paper] // ARM Limited. – 2013.

[46] *Hao S. et al.* Estimating Android applications' CPU energy usage via bytecode profiling // 2012 First international workshop on green and sustainable software (GREENS). – IEEE, 2012. – P. 1–7.

[47] *Hindle A. et al.* Greenminer: A hardware based mining software repositories software energy consumption framework // In: Proc. of the 11th

Working Conference on Mining Software Repositories. – 2014. – P. 12–21.

[48] *Hu Y. et al.* Lightweight energy consumption analysis and prediction for Android applications // Science of Computer Programming. – 2018. – Vol. 162. – P. 132–147.

[49] *Huang J. et al.* A close examination of performance and power characteristics of 4G LTE networks // Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services. – 2012. – P. 225–238.

[50] *Hung S. H. et al.* Performance and power estimation for mobile-cloud applications on virtualized platforms // Proceedings of the 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing. – IEEE, 2013. – P. 260–267.

[51] *Jung W., Kim K., Cha H.* Userscope: A fine-grained framework for collecting energy-related smartphone user contexts // 2013 International Conference on Parallel and Distributed Systems. – IEEE, 2013. – P. 158–165.

[52] *Kamiyama T., Inamura H., Ohta K.* A model-based energy profiler using online logging for Android applications // Proceedings of the 2014 Seventh International Conference on Mobile Computing and Ubiquitous Networking (ICMU). – IEEE, 2014. – P. 7–13.

[53] *Kapetanakis K., Panagiotakis S.* Efficient energy consumption's measurement on android devices // Proceedings of 2012 16th Panhellenic Conference on Informatics. – IEEE, 2012. – P. 351–356.

[54] Energy Aware Scheduling. Available at: `https://www.kernel.org/doc/html/next/scheduler/sched-energy.html`. (accessed 06.08.2023).

[55] Schedutil. Available at: `https://docs.kernel.org/scheduler/schedutil.html`. (accessed 12.08.2023)

[56] *Kiefer J., Wolfowitz J.* Statistical estimation on the maximum of a regression function // The Annals of Mathematical Statistics. – 1952. – Vol. 23. – P. 462–466.

[57] *Kim H. J., Kyong J., Lim S. S.* A systematic power and performance analysis framework for heterogeneous multiprocessor system // IEMEK Journal of Embedded Systems and Applications. – 2014. – Vol. 9, no. 6. – P. 315–321.

[58] *Kim S., Bin K., Ha S., Lee K., Chong S.* ZTT: learning-based DVFS with zero thermal throttling for mobile devices // In: Proc. of the 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '21). – 2021. – P. 41-–53.

[59] *Larsson M., Stigelid M.* Energy Efficient Data Synchronization in Mobile Applications. A Comparison Between Different Data Synchronization Techniques. – Ph.D. Thesis – 2015.

[60] *Lee J., Nam S., Park S.* Energy-Efficient Control of Mobile Processors Based on Long Short-Term Memory // IEEE Access. – 2019. – Vol. 7. – P. 80552-80560.

[61] *Lee S., Yoon C., Cha H.* User interaction-based profiling system for android application tuning // Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing. – 2014. – P. 289–299.

[62] *Lee S. et al.* EnTrack: a system facility for analyzing energy consumption of Android system services // Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing. – 2015. – P. 191–202.

[63] *Li D. et al.* Energy-directed test suite optimization // 2013 2nd International Workshop on Green and Sustainable Software (GREENS). – IEEE, 2013. – P. 62–69.

[64] *Li X., Gallagher J. P.* Fine-grained energy modeling for the source code of a mobile application // Proceedings of the 13th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services. – 2016. – P. 180–189.

[65] *Li X., Wen W., Wang X.* Usage history-directed power management for smartphones // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). – Springer. – 2015.

[66] *Marr D. T. et al.* Hyper-threading technology architecture and microarchitecture // Intel Technology Journal. – 2002. – Vol. 6, no. 1.

[67] Measuring Power Values. Available at: `https://source.android.com/docs/core/power/values`. (accessed 06.08.2023).

[68] *Metri G.C.* Energy Efficiency Analysis and Optimization for Mobile Platforms. – Wayne State University, 2014.

[69] *Metri G.C. et al.* A simplistic way for power profiling of mobile devices // In: Proc. of 2012 International Conference on Energy Aware Computing. — 2012. — P. 1–6.

[70] *Mikhalevich V. S., Gupal A. M., Norkin V. I.* Methods of non-convex optimization. – M.: Mir, 1987. – 279 p.

[71] *Mittal R., Kansal A., Chandra R.* Empowering developers to estimate app energy consumption // In: Proc. of the 18th Annual International Conference on Mobile Computing and Networking. – 2012. – P. 317–328.

[72] *Oliveira W. et al.* Recommending energy-efficient java collections // 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). – IEEE, 2019. – P. 160–170.

[73] *Myasnikov V., Sartasov S., Slesarev I., Gessen P.* Energy consumption measurement frameworks for Android OS: A systematic literature review // In: Proc. of the Fifth Conference on Software Engineering and Information Management (SEIM 2020). – 2020.

[74] 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 06.08.2023)

[75] *Ohk S. R., Kim Y. S., Kim Y. J.* Phase-based low power management combining CPU and GPU for Android smartphones // Electronics. — 2022. — Vol. 11, no. 16. — P. 2480.

[76] *Pandiyan D., Wu C. J.* Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms // Proceedings of 2014 IEEE International Symposium on Workload Characterization (IISWC). – IEEE, 2014. – P. 171–180.

[77] *Patros P. et al.* Toward sustainable serverless computing // IEEE Internet Computing. — IEEE, 2021. — Vol. 25, no. 6. — P. 42–50.

[78] *Pelogeiko M., Sartasov S., Granichin O.* On stochastic optimization for smartphone CPU energy consumption decrease // Informatics and Automation (SPIIRAS Proceedings). – 2023. – Vol. 22. – no. 5. – P. 1004–1033.

[79] *Polyak B. T., Tsybakov A. B.* Optimal Orders of Accuracy of Stochastic Optimization Search Algorithms // Problems of Information Transmission. – 1990. – Vol. 26. – № 2. – P. 45–53.

[80] *Polyak B. T., Tsybakov A. B.* On stochastic approximation with arbitrary noise (the KW case) / Topics in Nonparametric Estimation. Khasminskii R.Z. eds. // Advances in Soviet Mathematics. Amer. Math. Soc. Providence. – 1992. – Vol. 12. – P. 107–113.

[81] *Poornambigai K., Raj M. L., Meena P.* Reducing the energy consumption using dvfs performance optimizing scheme //EPRA International Journal of Research and Development (IJRD). — 2017. — Vol. 2. -— no. 1. — P. 79–88.

[82] Power Profiles for Android. Available at: `https://source.android.com/docs/core/power`. (accessed 06.08.2023).

[83] *Rapp M., Krohmer N., Khdr H., Henkel J.* NPU-accelerated imitation learning for thermal- and QoS-aware optimization of heterogeneous multi-cores // In: Proc. of the 2022 Conference & Exhibition on Design, Automation & Test in Europe (DATE '22). – 2021. – P. 584–587.

[84] *Robbins H., Monro S.* A stochastic approximation method // The Annals of Mathematical Statistics. – 1951. – Vol. 22. – P. 400–407.

[85] *Sahar H., Bangash A. A., Beg M. O.* Towards energy aware object-oriented development of android applications // Sustainable Computing: Informatics and Systems. – 2019. – Vol. 21. – P. 28–46.

[86] *Sahin C. et al.* Initial explorations on design pattern energy usage // 2012 First International Workshop on Green and Sustainable Software (GREENS). -— IEEE, 2012. – P. 55–61.

[87] *Saksonov A.* Method to Derive Energy Profiles for Android Platform. – Master Thesis. – 2014.

[88] *Sartasov S. Yu., Bogdanov E. A., Bozhnyuk A. S., Bykov D. V., Granichin O. N.* Problems and prospects of using stochastic approximation to control processor frequencies in Android OS // Computer tools in education. – 2021. – № 2. – P. 26–40.

[89] *Sartasov S. et al.* Navitas Framework: A Novel Tool for Android Applications Energy Profiling // In: Proc. of the Sixth Conference on Software Engineering and Information Management (SEIM 2021). – 2021.

[90] *Sartasov S., Miroshnikov V., Kuznetsov I.* Frequency-independent smartphone peripherals energy consumption estimation // Cybernetics and Physics. – 2023. – Vol. 12. – no. 1. – P. 42–50.

[91] *Sartasov S. Yu., Pelogeiko M. A., Granichin O. N.* Power saving DVFS governors for Android OS based on SPSA algorithm with two observations and OnDemand governor. – Certificate of registration of the computer program No. 2023666564 dated 02.08.23.

[92] *Shin D. et al.* Online estimation of the remaining energy capacity in mobile systems considering system-wide power consumption and battery characteristics // Proceedings of 2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC). – IEEE, 2013. – P. 59–64.

[93] *Song S., Kim J., Chung J.-M.* Energy consumption minimization control for augmented reality applications based on multi-core smart devices // In: Proc. of 2019 IEEE International Conference on Consumer Electronics (ICCE). – 2019. – P. 1–4.

[94] *Spall J.C.* Multivariate stochastic approximation using a simultaneous perturbation gradient approximation // IEEE Transactions on Automatic Control. – 1992. – Vol. 37, no. 3. – P. 332–341.

[95] *Spall J.C.* A one measurement form of simultaneous perturbation stochastic approximation // Automatica. – 1997. – Vol. 33. – P. 109–112.

[96] *Spall J. C.* An overview of the simultaneous perturbation method for efficient optimization / Johns Hopkins apl technical digest. – 1998. – Vol. 19. – No. 4. – P. 482–492.

[97] *Tsao S. L. et al.* Powermemo: A power profiling tool for mobile devices in an emulated wireless environment // Proceedings of 2012 International Symposium on System on Chip (SoC). – IEEE, 2012. – P. 1–5.

[98] *Tuysuz M. F., Ucan M., Trestian R.* A real-time power monitoring and energy-efficient network/interface selection tool for android smartphones // Journal of Network and Computer Applications. – 2019. –Vol. 127. – P. 107–121.

[99] *Vakhitov A. T., Granichin O. N.* Randomized estimation algorithms for irregular noise // Stochastic optimization in informatics. – 2006. – Vol. 2. – C. 3–37.

[100] *Vakhitov A. T., Granichin O. N.* SPSA-based adaptive control: accuracy of estimates // IFAC Proceedings Volumes. – 2007. – Vol. 40, no. 13. – P. 429–434.

[101] *Vazan M.* Stochastic approximation. – M.: Mir, 1972. – 295 p.

[102] *Walcott-Justice K.* MAUE: A Framework For Detecting Energy Bugs From User Interactions On Mobile Applications. – Master Thesis. – 2016.

[103] *Weiser M., Welch B., Demers A., Shenker S.* Scheduling for reduced CPU energy // Mobile Computing. – Springer. – 1996. – P. 449–471.

[104] *Westfield B., Gopalan A.* Orka: A new technique to profile the energy usage of Android applications // In: Proc. of 2016 5th International Conference on Smart Cities and Green ICT Systems (SMARTGREENS). – IEEE, 2016. – P. 1–12.

[105] *Wilke C., Götz S., Richly S.* Jouleunit: a generic framework for software energy profiling and testing // Proceedings of the 2013 Workshop on Green in/by Software Engineering. – 2013. – P. 9–14.

[106] *Wilke C.* Energy-Aware Development and Labeling for Mobile Applications. – Ph.D. Thesis. – 2014.

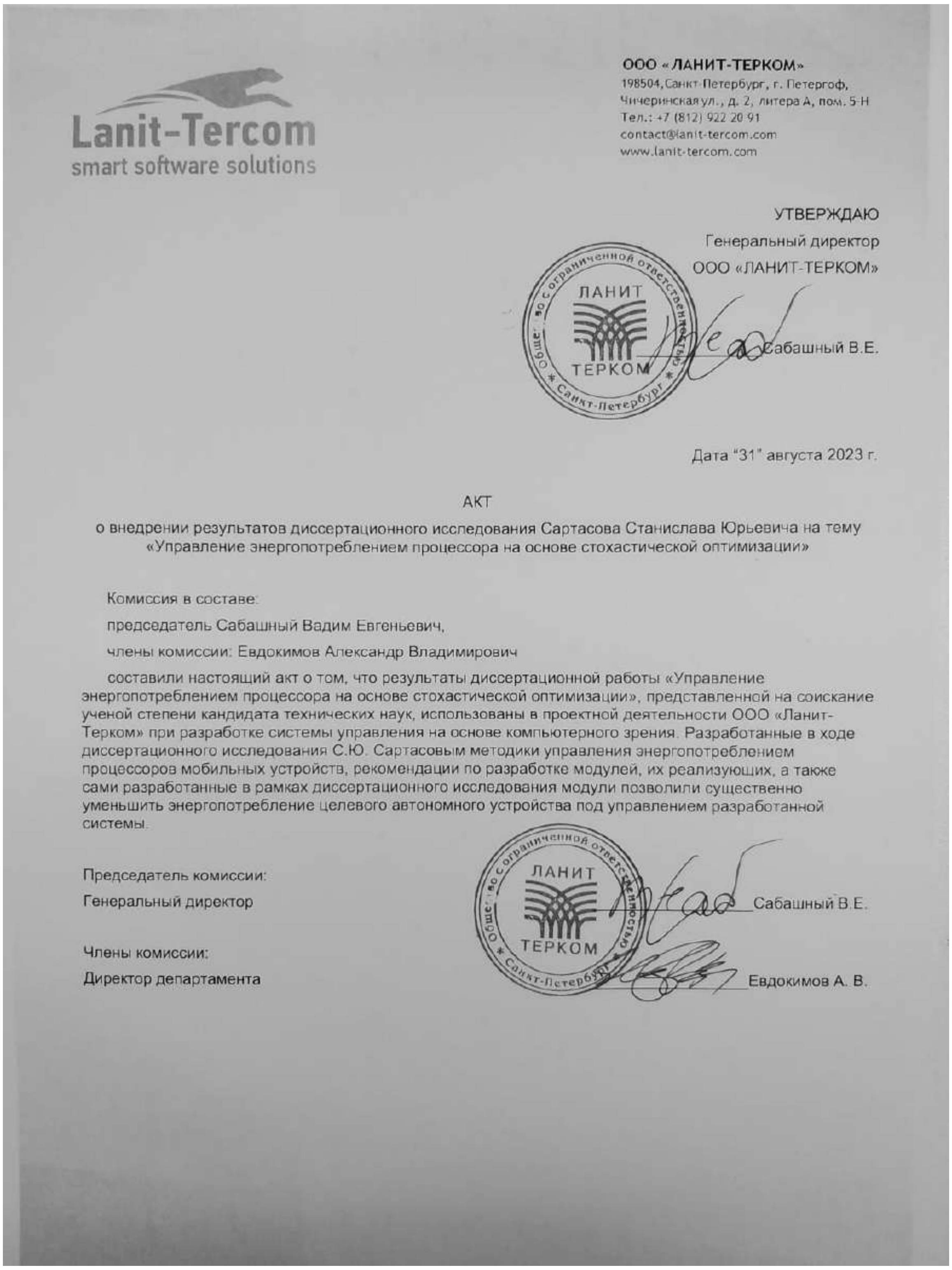
[107] *Wu H., Sun Y., Wolter K.* Energy-efficient decision making for mobile cloud offloading // IEEE Transactions on Cloud Computing. — IEEE, 2018. — Vol. 8, no. 2. – P. 570–584.

[108] *Wysocky R.J.* CPU Performance Scaling. Available at: `https://www.kernel.org/doc/html/latest/admin-guide/pm/cpufreq.html`. (accessed 06.08.2023).

[109] *Yoon C. et al.* AppScope: Application energy metering framework for Android smartphone using kernel activity monitoring // Proceedings of the 2012 USENIX Annual Technical Conference (USENIX ATC 12). – 2012. – P. 387–400.

[110] *Zhang L. et al.* Accurate online power estimation and automatic battery behavior based power model generation for smartphones // Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. – 2010. – P. 105–114.

Appendix A. Implementation acts



Lanit-Tercom
smart software solutions

ООО «ЛАНИТ-ТЕРКОМ»
198504, Санкт-Петербург, г. Петергоф,
Чичеринская ул., д. 2, литера А, пом. 5-Н
Тел.: +7 (812) 922 20 91
contact@lanit-tercom.com
www.lanit-tercom.com

УТВЕРЖДАЮ
Генеральный директор
ООО «ЛАНИТ-ТЕРКОМ»

Общество с ограниченной ответственностью * Санкт-Петербург * ЛАНИТ ТЕРКОМ

[illegible] Сабашный В.Е.

Дата "31" августа 2023 г.

АКТ

о внедрении результатов диссертационного исследования Сартасова Станислава Юрьевича на тему «Управление энергопотреблением процессора на основе стохастической оптимизации»

Комиссия в составе:

председатель Сабашный Вадим Евгеньевич,

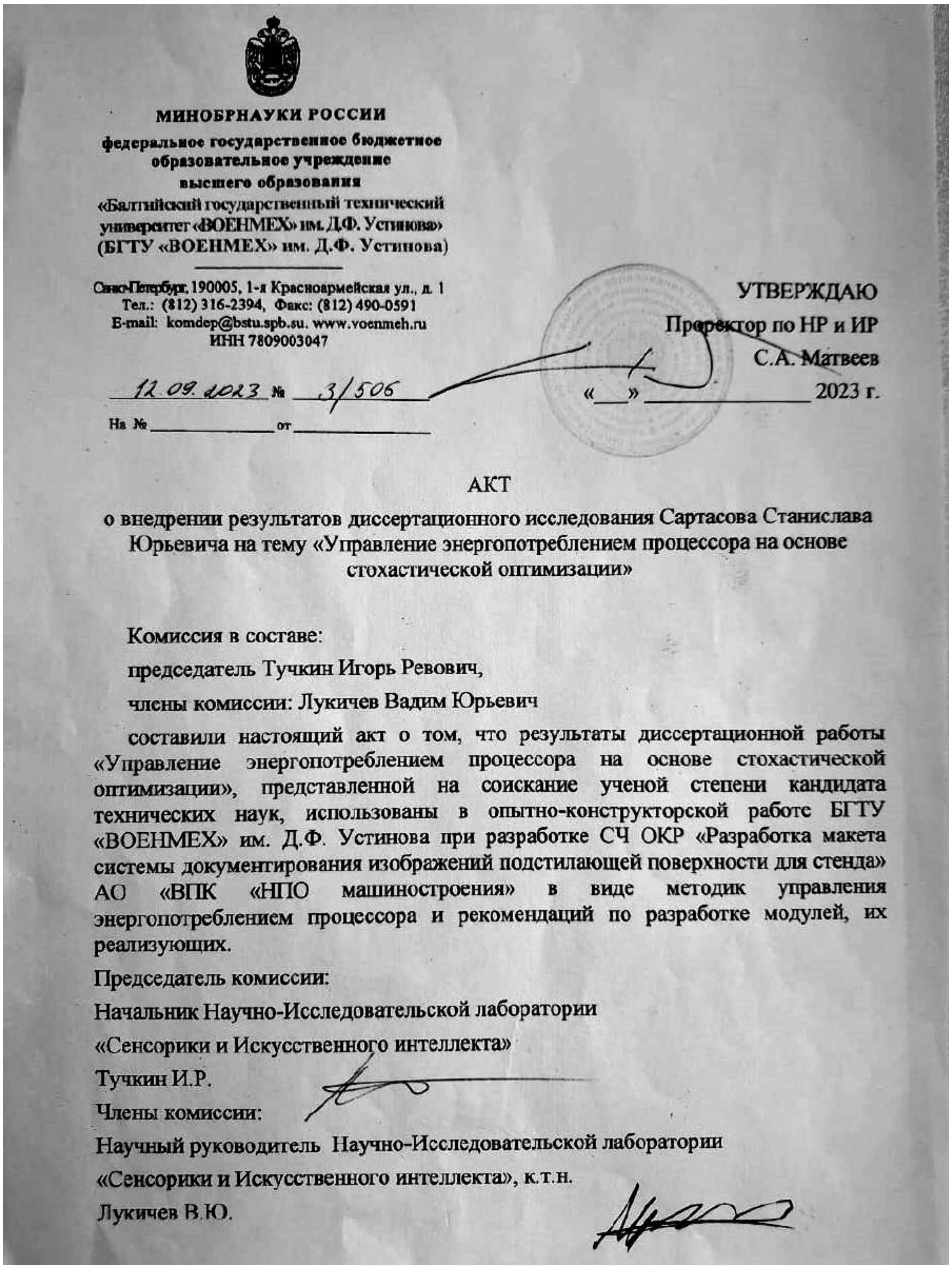
члены комиссии: Евдокимов Александр Владимирович

составили настоящий акт о том, что результаты диссертационной работы «Управление энергопотреблением процессора на основе стохастической оптимизации», представленной на соискание ученой степени кандидата технических наук, использованы в проектной деятельности ООО «Ланит-Терком» при разработке системы управления на основе компьютерного зрения. Разработанные в ходе диссертационного исследования С.Ю. Сартасовым методики управления энергопотреблением процессоров мобильных устройств, рекомендации по разработке модулей, их реализующих, а также сами разработанные в рамках диссертационного исследования модули позволили существенно уменьшить энергопотребление целевого автономного устройства под управлением разработанной системы.

Председатель комиссии:
Генеральный директор [illegible] Сабашный В.Е.

Общество с ограниченной ответственностью * Санкт-Петербург * ЛАНИТ ТЕРКОМ

Члены комиссии:
Директор департамента [illegible] Евдокимов А. В.



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное
образовательное учреждение
высшего образования
«Балтийский государственный технический
университет «ВОЕНМЕХ» им. Д.Ф. Устинова»
(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова)

Санкт-Петербург, 190005, 1-я Красноармейская ул., д. 1
Тел.: (812) 316-2394, Факс: (812) 490-0591
E-mail: komdep@bstu.spb.su. www.voenmeh.ru
ИНН 7809003047

12.09.2023 № 3/506
На № ______________ от ______________

УТВЕРЖДАЮ
Проректор по НР и ИР
С.А. Матвеев
«___» ______________ 2023 г.

АКТ

о внедрении результатов диссертационного исследования Сартасова Станислава Юрьевича на тему «Управление энергопотреблением процессора на основе стохастической оптимизации»

Комиссия в составе:

председатель Тучкин Игорь Ревович,

члены комиссии: Лукичев Вадим Юрьевич

составили настоящий акт о том, что результаты диссертационной работы «Управление энергопотреблением процессора на основе стохастической оптимизации», представленной на соискание ученой степени кандидата технических наук, использованы в опытно-конструкторской работе БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова при разработке СЧ ОКР «Разработка макета системы документирования изображений подстилающей поверхности для стенда» АО «ВПК «НПО машиностроения» в виде методик управления энергопотреблением процессора и рекомендаций по разработке модулей, их реализующих.

Председатель комиссии:

Начальник Научно-Исследовательской лаборатории

«Сенсорики и Искусственного интеллекта»

Тучкин И.Р. [illegible]

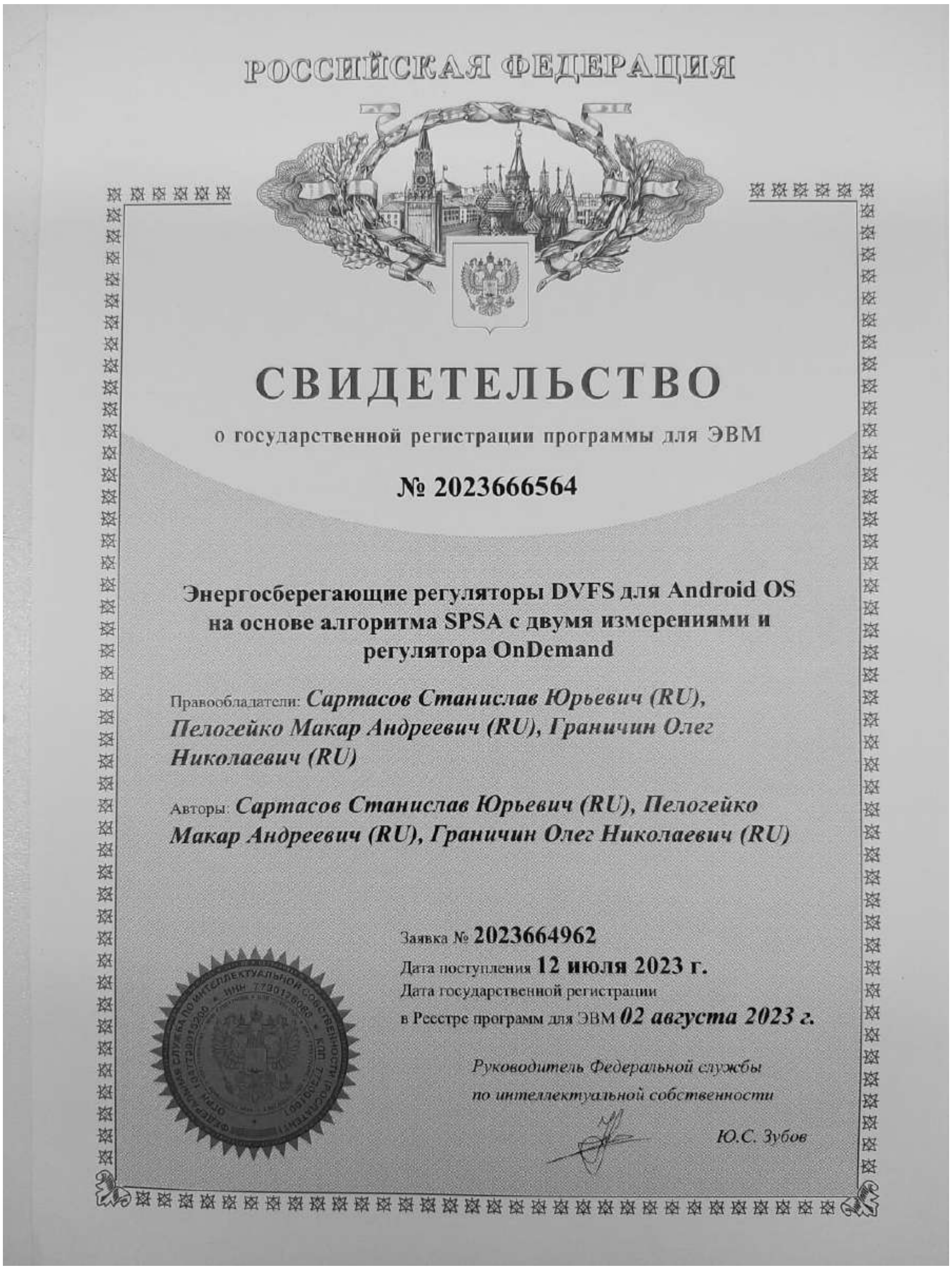
Члены комиссии:

Научный руководитель Научно-Исследовательской лаборатории

«Сенсорики и Искусственного интеллекта», к.т.н.

Лукичев В.Ю. [illegible]

Appendix B. Certificate of state registration of a computer program



РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2023666564

Энергосберегающие регуляторы DVFS для Android OS на основе алгоритма SPSA с двумя измерениями и регулятора OnDemand

Правообладатели: ***Сартасов Станислав Юрьевич (RU), Пелогейко Макар Андреевич (RU), Граничин Олег Николаевич (RU)***

Авторы: ***Сартасов Станислав Юрьевич (RU), Пелогейко Макар Андреевич (RU), Граничин Олег Николаевич (RU)***

Заявка № **2023664962**

Дата поступления **12 июля 2023 г.**

Дата государственной регистрации

в Реестре программ для ЭВМ ***02 августа 2023 г.***

Руководитель Федеральной службы по интеллектуальной собственности

[illegible]

Ю.С. Зубов

ФЕДЕРАЛЬНАЯ СЛУЖБА ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ (РОСПАТЕНТ) • ИНН 7730176088 • КПП 773001001 • ОГРН 1047730015200