Санкт-Петербургский государственный университет

Пелогейко Макар Андреевич

Выпускная квалификационная работа

Энергосбережение в Android OS путём динамического изменения частоты процессора

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2019 «Программная инженерия»*

Научный руководитель:
д. ф.-м. н., профессор О. Н. Граничин

Рецензент:
к. ф.-м. н., доцент Ю. В. Иванский

Санкт-Петербург
2023

Saint Petersburg State University

Makar Pelogeiko

Bachelor's Thesis

Power saving in Android OS by dynamically changing the processor frequency

Education level: bachelor

Speciality *09.03.04 "Software Engineering"*

Educational Programme *CB.5080.2019 "Software Engineering"*

Scientific supervisor:
Sc.D, prof. O.N. Granichin

Reviewer:
C.Sc., docent Y.V. Ivansky

Saint Petersburg
2023

Оглавление

Введение

С каждым годом смартфоны становятся более важной частью жизни людей, а их количество продолжает расти. Вместе с этим увеличивается уровень технических возможностей, и постоянно появляются решения для увеличения времени автономной работы смартфонов.

Во время эксплуатации смартфона возникают ситуации, когда некоторые задачи не требуют большого количества вычислительных ресурсов. При правильном соотношении производительности и энергопотребления в текущий момент времени можно увеличить время автономной работы устройства и при этом сохранить комфортную для пользователя производительность.

В этой работе используется ОС Android по причине того, что Android является одной из трех наиболее распространенных операционных систем для мобильных устройств и имеет открытый исходный код.

Существует подход к управлению частотой и напряжением процессора — Dynamic Voltage and Frequency Scaling (DVFS). Он позволяет оптимизировать энергопотребление и производительность процессора.

В ОС Android имеется система CPUFreq, которая позволяет использовать алгоритмы динамического масштабирования напряжения и частоты (DVFS), исходя из тех или иных критериев оптимизации и стратегий их достижения.

В настоящее время активно исследуются вопросы, связанные с разработкой DVFS-регуляторов. Существует множество подходов их реализации. Так как нагрузку на процессор трудно предсказать и физические средства наблюдения вносят помехи в измерения, стохастические алгоритмы становятся всё более популярными. SPSA (Simultaneous perturbation stochastic approximation) [17] является одним из таких стохастических алгоритмов, который может использоваться для решения задачи трекинга оптимальной точки с почти произвольными помехами.

Ранее был разработан DVFS-регулятор, основанный на SPSA с одним измерением [9]. Данная работа посвящена разработке

DVFS-регулятора, основанного на алгоритме SPSA с двумя измерениями, и его апробации на реальном устройстве.

1. Постановка задачи

Целью данной работы является разработка DVFS-регулятора на основе алгоритма SPSA с двумя измерениями и сравнение его с существующими аналогами на реальном смартфоне.

Для достижения данной цели были сформулированы следующие задачи:

1. Провести обзор существующих DVFS-регуляторов.
2. Подготовить тестовый стенд для сравнения DVFS-регуляторов.
3. Создать методику тестирования для сравнения DVFS-регуляторов.
4. Разработать DVFS-регулятор на основе SPSA с двумя измерениями и встроить его в ОС Android.
5. Разработать модифицированную версию DVFS-регулятора на основе SPSA с двумя измерениями и интегрированную с EAS и встроить его в ОС Android.
6. Провести тестирование полученных регуляторов.

2. Обзор предметной области

2.1. Система CPUFreq

CPUFreq это система для изменения частоты ядер процессора. Она состоит из 3-х частей [16]:

1. Ядро — базовая инфраструктура и программные интерфейсы для выбора частот ядер процессора. Эта система динамически указывает драйверам частоту, на которую должны быть переведены определенные ядра процессора посредством DVFS-регуляторов (scaling governors).

2. Драйвер — модуль, осуществляющий передачу решений, полученных от DVFS-регуляторов, в аппаратную часть устройства.

3. Регулятор — модуль, содержащий алгоритм выбора частоты для ядер процессора. Следует отметить, что решение, полученное от регулятора, имеет рекомендательный характер и может быть применено не во всех ситуациях.

Стоит отметить, что переключение на новую частоту занимает некоторое время. Например, для тестового стенда на базе Xiaomi Redmi Note 8 это 30 мсек. Система CPUFreq не гарантирует того, что основная процедура регулятора не будет вызвана до завершения смены частоты, вызванной предыдущим решением регулятора.

2.2. Стандартные регуляторы частоты

В этом разделе представлено описание стандартных DVFS-регуляторов от простых к более сложным.

Powersave — рекомендует выбирать минимальную частоту, чтобы значительно увеличить время автономной работы устройства.

Performance — рекомендует выбирать максимальную частоту, чтобы достичь высокой производительности устройства.

Userspace — дает возможность пользователю или программе самостоятельно выбирать частоту для кластеров процессора.

OnDemand — один из наиболее распространенных DVFS-регуляторов, который является основой для более современных подходов к управлению частотами ядер процессора. Для определения целевой частоты этот DVFS-регулятор использует нагрузку на процессор в качестве показателя. Нагрузка кластера равна максимальной нагрузке среди всех его ядер. Нагрузка ядра вычисляется путем измерения времени, в течении которого ядро находилось в активном состоянии, между вызовами рабочей процедуры DVFS-регулятора, и вычисления отношения этого времени к общему времени, которое представляет собой сумму активного времени и времени простоя. Частота выбирается пропорционально этой нагрузке, за исключением случаев, когда нагрузка превышает порог, указанный в параметре up_threshold. В таких случаях целевой частотой становится самая высокая частота из доступных. Главным недостатком этого DVFS-регулятора являются резкие скачки частоты процессора.

Conservative — DVFS-регулятор, который аналогично DVFS-регулятору OnDemand использует нагрузку на процессор. За счет изменения частоты небольшими шагами, указанными в параметрах freq_step и sampling_down_factor, этот DVFS-регулятор избегает резких скачков частоты за короткое время.

Interactive — DVFS-регулятор, который был создан для устройств, где требуется сократить задержки при работе. Регуляторы OnDemand и Conservative вызываются с определенной периодичностью, из-за чего частота процессора между этими вызовами может быть не оптимальной. Interactive же вызывается не только периодически, но и при выходе

ядер процессора из простоя. Когда некоторое ядро процессора выходит из состояния простоя, происходит измерение нагрузки на кластер этого ядра в течении времени, указанного в параметре min_sample_time. Далее частота выбирается пропорционально нагрузке или повышается до максимальной, если нагрузка превысила порог, указанный в параметре go_hispeed_load. Такой подход позволяет быстро реагировать на повышение нагрузки на процессор.

2.3. Современные регуляторы частоты

В данном разделе представлен обзор некоторых DVFS-регуляторов, разработанных за последнее время.

В работе [7] DVFS-регулятор базируется на регуляторе OnDemand. Разрабатываемый авторами регулятор дополнен функциональностью переводить ядра в режим сна. Суть модификации состоит в том, чтобы переводить ядро в сон при нагрузке меньше чем 10% и далее производить пересчет частот для ядер по формуле:

$$f_{new} = \frac{n_{max} * f_{OnDemand} * K}{n} \quad (1)$$

где f_{new} — новая частота для данного ядра, n_{max} — количество ядер процессора, $f_{OnDemand}$ — решение регулятора OnDemand для данного ядра, K — общая нагрузка на смартфон, измеряемая в процентах, n — количество активных ядер. Авторы показывают, что в сравнении со стандартным регулятором Android их подход при тестировании с использованием тяжелых вычислительных приложений показал наилучший результат — 11.7% энергосбережения относительно стандартной конфигурации Android ОС на их тестовом стенде.

В работе [8] авторы предлагают учитывать то, какие инструкции выполняются. Все возможные состояния процессора были поделены на фазы в зависимости от того, насколько используется процессор и каким образом процессор взаимодействует с памятью в данный момент. Для предсказания того, в какой фазе процессор будет находится в следующий момент времени, используется взвешенная статистика недавнего

использования. Далее вычисляется энергопотребление и частота в соответствии с равенством:

$$P_n = P_{old} * \frac{V_n * f_n}{V_{old} * f_{old}} \tag{2}$$

где P_n — новая потребляемая мощность, P_{old} — мощность, потребляемая на предыдущем шаге, V_n — новое напряжение ядра, f_n — новая частота ядра, V_{old} — напряжение ядра на предыдущем шаге, f_{old} — частота ядра на предыдущем шаге. Как показывают авторы, данный подход позволил снизить энергопотребление примерно на 18%.

В статье [9] на основе DVFS-регулятора OnDemand был разработан регулятор с использованием алгоритма SPSA с 1 измерением. Был выведен функционал качества:

$$F_\tau(f) = 2^{\frac{(workload_\tau(f)-\lambda)}{2}} + \gamma 1.5^{table(f)} \tag{3}$$

где τ — период времени, $workload_\tau(f)$ — нагрузка за период времени τ (принимает значение от 0 до 1), λ — эмпирически выверенный порог нагрузки, γ — параметр, отвечающий за чувствительность системы к используемой частоте, $table(f)$ — порядковый номер частоты f в таблице частот процессора. Далее, следуя формуле SPSA с 1 измерением, этот функционал использовался в вычислениях оптимальной частоты ядер процессора. В результате получился регулятор, не уступающий по эффективности таким популярным регуляторам, как Interactive и OnDemand.

2.4. Обзор системы EAS

Energy Aware Scheduling (EAS) — это модификация стандартного планировщика Completely Fair Scheduler (CFS), при помощи которой планировщик может оценивать влияние своих решений на энергопотребление ядер процессора. EAS использует модель энергопотребления (Energy Model, EM) ядер для назначения каждой задачи на некоторое ядро так, чтобы сократить энергопотребление процессора в целом

с минимальным влиянием на производительность [15]. Вместе с этим EM должна быть достаточно простой для минимизации времени работы планировщика.

EAS создан для работы с гетерогенными архитектурами ЦП (например, Arm big.LITTLE [1, 12]), поскольку в таких архитектурах есть возможность сократить энергопотребление за счет планирования.

Цель EAS — минимизировать энергопотребление и выполнить программную задачу, при этом сохраняя хорошую производительность. Это является формой оптимизации, отличной от цели планировщика, который ориентирован только на повышение производительности. Вместо этого EAS учитывает две цели: энергоэффективность и производительность.

Таким образом, EAS предлагает новый способ назначения задач ядрам. При планировании задачи используется EM для выбора ядра, которое обеспечит необходимые вычислительные мощности для планируемой задачи при минимальном приросте энергопотребления. Прогнозы EAS основаны на знании о топологии платформы и учитывают возможности производительности ядер и соответствующие им энергопотребление.

2.5. Обзор регулятора schedutil

Schedutil — это DVFS-регулятор, который был разработан специально для EAS. Он дает возможность управлять выбором частоты процессора с учетом потребностей планировщика. Schedutil можно рассматривать как посредника между планировщиком и CPUFreq (см. Рис. 1). Это позволяет планировщику реализовать алгоритм выбора частоты в каком-то смысле самостоятельно. Такая связь между планировщиком и DVFS-регулятором дает возможность уточнять общесистемные политики, что приводит к уменьшению энергопотребления с малым ущербом производительности [2].

Schedutil имеет пороги регулирования для предотвращения частой смены частоты (up_threshold и down_threshold). Также за счет этих

порогов имеется возможность увеличить частоту в случае резкого повышения нагрузки и корректно обработать ситуации при кратковременных падениях нагрузки.

Стоит отметить, что Schedutil имеет сходство с DVFS-регулятором OnDemand. Оба DVFS-регулятора используют нагрузку на процессор для выбора частоты. Их основные отличия состоят в том, что Schedutil имеет дополнительную функциональность, связанную с планировщиком, и в том, что Schedutil использует отличный от OnDemand способ получения нагрузки на рассматриваемое ядро, связанный с EAS.

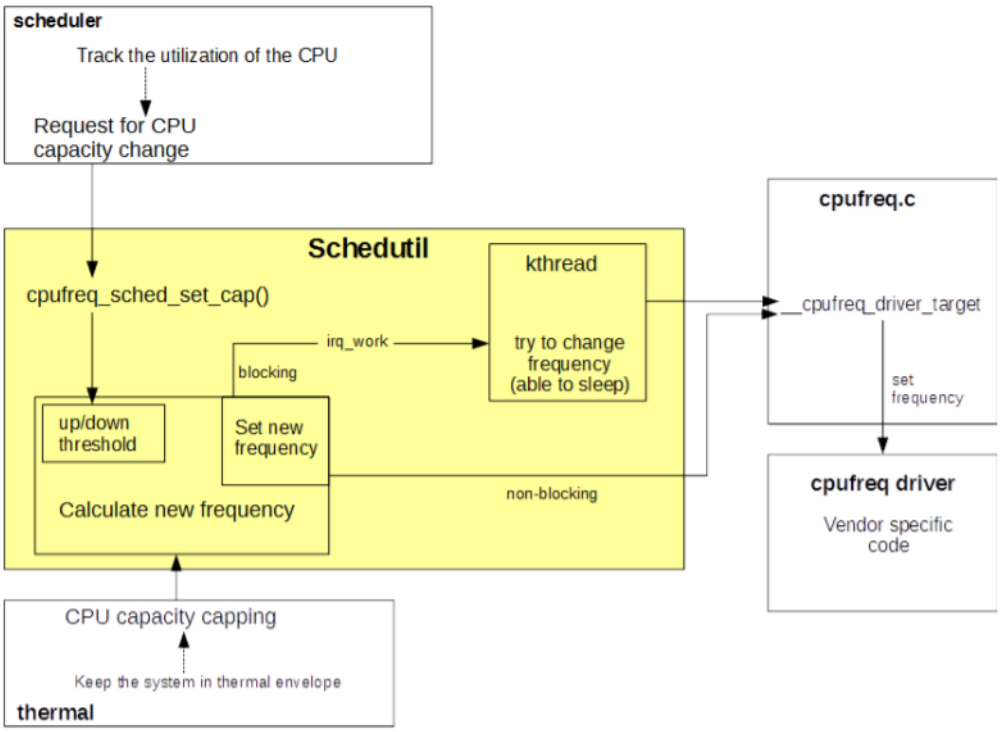

Рис. 1: Структура Schedutil [2].

2.6. Алгоритмы стохастической аппроксимации со случайными направлениями

Рассмотрим задачу трекинга, в которой необходимо найти некоторое со временем дрейфующее оптимальное значение параметра $x \in \mathbb{R}^p$ в

дискретный момент времени n. Пусть имеется возможность оценивать текущее значение искомого параметра и изменять его значение с помехами. Тогда обозначим некоторый функционал качества для оценки текущего значения искомого параметра как
$F(x_n, w_n) : \mathbb{R}^p \times \mathbb{R}^q \to \mathbb{R}^1$, где n — дискретный момент времени ($n \in [0; \infty)$), x_n — параметр из $\mathbb{R}^p$, оптимальное значение которого необходимо найти, w_n — случайный вектор помех из $\mathbb{R}^q$, пораженный распределением $P_w(\cdot)$. Таким образом, возможно оценить функционал среднего риска, как $f(x_n) = E_w\{F(x_n, w_n)\}$, где $E_w\{\cdot\}$ — математическое ожидание по $P_w(\cdot)$.

При измерении значений $F(x_n, w_n)$ имеет место аддитивная центрированная независимая случайная ошибка $v_n \in \mathbb{R}$. В таком случае измерение имеет следующий вид: $y_n = f(x_n) + v_n$.

Для решения задачи используется случайный вектор Δ_n, координаты которого имеют значения ± 1 с одинаковой вероятностью и не зависят друг от друга.

Таким образом, задача трекинга может решается при помощи следующих алгоритмов [17]:

$$\hat{\theta}_{n+1} = \hat{\theta}_n - \frac{\alpha_n(y_n^+ - y_n^0)}{\Delta_n \beta_n} \tag{4}$$

$$\hat{\theta}_{n+1} = \hat{\theta}_n - \frac{\alpha_n(y_n^+ - y_n^-)}{2\Delta_n \beta_n} \tag{5}$$

где

$$y_n^0 = f(\hat{\theta}_n) + v_n^0,\ y_n^+ = f(\hat{\theta}_n + \Delta_n\beta_n) + v_n^+,\ y_n^- = f(\hat{\theta}_n - \Delta_n\beta_n) + v_n^-,$$

$\{\alpha_n\}$ и $\{\beta_n\}$ — последовательности неотрицательных чисел, $\{\hat{\theta}_n\}$ — последовательность оценок искомого параметра.

Алгоритм (4) в англоязычной литературе получил название Simultaneous perturbation stochastic approximation (SPSA), а в русскоязычной — рандомизированный алгоритм стохастической аппроксимации. Алгоритм (5) является его вариацией с двумя измерениями.

Следует отметить, что для решения задачи минимизации функци-

онала среднего риска $\{\alpha_n\}$ и $\{\beta_n\}$ являются последовательностями положительных чисел, стремящимися к нулю, но для решения задачи задачи трекинга можно использовать константные значения для α и β [6].

3. Подготовка тестового стенда

3.1. Описание тестового стенда SM-G930F

В качестве тестового стенда был взят Samsung
Galaxy s7 SM-G930F [13] 2016 года выпуска. Характеристики:

- процессор: Exynos 8 Octa (8890) в котором 4 ядра Cortex-A53 (LITTLE) и 4 ядра Exynos M1 (big) [3];
- ОС: Android 8 (Oreo);
- ОЗУ: 4ГБ;
- ПЗУ: 32ГБ.

Данный смартфон был выбран в качестве тестового стенда, так как имелся в наличии, допускал установку современных версий Android, а именно Android 10 и Android 11, и использовал процессор с архитектурой ARM big.LITTLE.

3.2. Необходимые изменения тестового стенда

В первую очередь на смартфоне был разблокирован загрузчик и установлена утилита восстановления TWRP[1]. Далее при помощи
TWRP была установлена операционная система lineage os 18.1 (Android 11)[2]. Следующим шагом были получены root права при помощи проектов Magisk[3] и no-verity-opt-encrypt[4]. После этого был осуществлен поиск исходного кода ядра, подходящего под полученную конфигурацию смартфона и имеющего подсистему EAS. Проект herolte[5] подошел по данным критериям. Данный проект был скомпилирован при помщи gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu[6] с установленными

[1] https://twrp.me/
[2] https://lineageos.org/
[3] https://magisk.me/
[4] https://build.nethunter.com/android-tools/no-verity-opt-encrypt/
[5] https://github.com/pascua28/herolte
[6] https://releases.linaro.org/components/toolchain/binaries/7.4-2019.02/aarch64-linux-gnu/

отладочными флагами конфигурации. Установочный zip-файл был получен при помощи lazyflasher[7]. Поскольку данное ядро имеет незначительную неисправность в настройках смартфона параметр `Система > Для разработчиков > Конфигурация USB по умолчанию` устанавливался как "`Без передачи данных`", также в этом разделе настроек были включены параметры: `Отладка по USB` и `Отладка суперпользователем`. Далее при помощи Android debug bridge (ADB) [5] установочный zip-файл загружался на смартфон и устанавливался через TWRP.

Для тестирования без EAS использовалась операционная система lineage os 17.1 (Android 10)[8] с ядром android_kernel_samsung_universal8890[9]. Для компиляции ядра использовался android-ndk-r12b/toolchains/aarch64-linux-android-4.9/prebuilt/linux-x86_64/bin/aarch64-linux-android-[10]. Процессы установки ОС, сборки и установки ядра аналогичны описанному ранее.

[7] https://github.com/makar-pelogeiko/lazyflasher

[8] https://lineageos.org/

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

[10] https://github.com/android/ndk/wiki/Unsupported-Downloads

4. Разработка методики для сравнения DVFS-регуляторов

4.1. Инструменты для сравнения DVFS-регуляторов

Чтобы оценить энергопотребление процессора, использующего определенный DVFS-регулятор, было решено разработать автоматизированные воспроизводимые тесты, отражающие поведение смартфона в реальных условиях. Для их создания был необходим инструмент, который позволяет управлять тестируемым смартфоном, имитируя действия пользователя. Эту задачу решает ADB [5]. Также с помощью данной технологии можно решать другие задачи тестирования: передача информации с тестирующего ПК на смартфон, передача информации со смартфона на ПК, установка *.apk файлов, запуск конкретных приложений на смартфоне, root доступ к консоли смартфона для настройки параметров системы.

Для сравнения производительности было решено использовать бенчмарк Geekbench 5.5.1[11] [11], так как предоставляемые этим бенчмарком данные о тестировании удобны для анализа и находятся в интернете в открытом доступе.

4.2. Определение способов измерения энергопотребления

Существуют различные подходы оценки энергопотребления процессора смартфона. Одиним из самых дешевых, быстрых и релевантных способов в нашем случае является оценка потребляемой мощности ядром процессора (P), как произведение текущей частоты (f) на напряжение (u) $P \sim fu^2$ [14].

Первый подход оценки энергопотребления процессора в данной работе состоит в том, чтобы через ADB [5] получить количество времени

[11] https://www.geekbench.com/

в 10-миллисекундных интервалах, проведенного каждым ядром процессора на доступных частотах [10], умножить полученные значения на соответствующие константы энергопотребления, предоставленные производителем, и сложить все произведения. После этого полученный результат переводится в мА·ч. Для этого время переводится из 10-миллисекундных интервалов в часы путем деления на 3600·100.

Однако не все ядра процессора используются постоянно. При отсутствии задач свободные ядра процессора переводятся в состояния простоя (c-state), в котором их энергопотребление ниже [4]. Также стоит отметить, что существует множество состояний простоя, и все они подчиняются правилу: чем глубже состояние простоя, тем меньше энергопотребление находящегося в нем ядра, а нулевое состояние простоя (C0) имеет энергопотребление, приблизительно равное рабочему энергопотреблению ядра на данной частоте.

Учитывая данную особенность и то, что у ядер процессора тестового стенда (SM-G930F) всего 2 состояния простоя: C0 и C1, было принято решение использовать еще 2 подхода оценки энергопотребления процессора. В следующем подходе время, проведенное ядром в состоянии C1, вычиталось из времени, проведенного этим ядром на минимальной доступной частоте, а далее использовались выше описанные вычисления. Поскольку нет гарантии того, что ядро простаивало только на минимальной частоте, данный подход не является точным. В заключительном подходе время, проведенное ядром в состоянии C1, вычиталось из времен на всех частотах пропорционально тому, сколько времени ядро провело на каждой частоте. Следует отметить, что данный подход тоже не является точным, но даёт достаточно хорошую с эмпирической точки зрения оценку.

4.3. Определение сценариев тестирования

Для адекватной оценки энергопотребления смартфона необходимо использовать несколько сценариев тестирования, приближенных к сценариям использования смартфона в реальной жизни и по-разному на-

гружающих процессор. Таким образом, был получен следующий список сценариев тестирования:

1. **videoVLC** — воспроизведение *.mp4 видео файла при помощи VLC плеера. VLC плеер должен быть выбран плеером по умолчанию.

2. **trialXTreme3** — запуск требовательной к производительности игры и имитация ее прохождения.

3. **flappyBird** — запуск flappyBird менее требовательной игры и имитация ее прохождения.

4. **type** — запуск приложения заметок, создание заметки, набор текста, сохранение заметки, удаление заметки.

5. **camera** — запуск приложения съмки видео, запись видео, удаление полученного видеофайла.

6. **twitch** — запуск браузера, переход по ссылке на стрим на платформе twitch, просмотр стрима, выход из браузера.

4.4. Особенности реализации

Для реализации инструмента тестирования был выбран язык Python ввиду его популярности и кросплатформенности. Инструмент тестирования[12] был реализован в виде нескольких утилит:

- утилита тестирования;
- утилита смены текущего DVFS-регулятора;
- утилита создания графиков по полученным данным;
- утилита для проведения тестирования нескольких DVFS-регуляторов и построения графиков по полученным данным.

[12] https://github.com/makar-pelogeiko/freq_gov_test

Такой подход дал возможность проводить как короткие тесты в процессе разработки DVFS-регулятора, так и более длинные тесты для сравнения большого количества DVFS-регуляторов.

Для возможности быстрого добавления тестовых сценариев каждый сценарий реализуется в определенном классе и динамически подключается к проекту тестирования.

Для возможности быстрой и гибкой настройки параметров проводимого тестирования основные настройки помещены в файл `config.py`.

5. Создание регулятора частоты

5.1. Выбор функционала качества

Для создания функционала качества использовалось понятие эффективности исполнения или cost of execution:

$$cost\_of\_execution_\tau(i) = \frac{energy\_const_\tau(i)}{freq_\tau(i)} \tag{6}$$

Здесь i — индекс частоты кластера τ в упорядоченном по возрастанию массиве, $energy\_const_\tau(i)$ — константа энергопотребления соответствующая частоте с индексом i кластера τ, $freq_\tau(i)$ — частота с индексом i кластера τ.

Таким образом, была создана модель, позволяющая оценить, отклонение текущей частоты кластера ядра от теоретически оптимальной:

$$|curr\_i_\tau - result\_i_\tau| \tag{7}$$

Здесь τ — кластер ядра процессора, $curr\_i_\tau$ — индекс текущей частоты, $result\_i_\tau$ — индекс теоретической оптимальной частоты, который вычисляется следующим образом:

$$result\_i_\tau = \operatorname*{argmin}_{\forall i \in \Phi} cost\_of\_execution_\tau(i) \tag{8}$$

Здесь $\Phi = \{\forall i : i \in FREQ_\tau \mid load_\tau(i) \leq target\_load_\tau\}$, $FREQ_\tau$ — множество всех доступных индексов частоты кластера τ, $load_\tau(i)$ — предполагаемая нагрузка на кластер τ при использовании частоты i,
$target\_load_\tau$ — целевая нагрузка на кластер τ (задается в виде параметра алгоритма), при превышении целевой нагрузки на кластер предлагается повышать частоту так, чтобы нагрузка снова стала меньше целевой.

5.2. Реализация DVFS-регуляторов на основе SPSA с двумя измерениями

В процессе изучения смартфона было обнаружено, что нагрузка на отдельные ядра и на весь процессор в целом быстро меняется. Вдобавок существует задержка между вызовом DVFS-регулятора и фактической сменой частоты кластера процессора. Поэтому для быстрой и точной реакции на изменение нагрузки на ядра процессора необходимо минимизировать количество вызовов DVFS-регулятора. Для решения этой задачи было реализовано несколько DVFS-регуляторов.

Для каждого кластера процессора в системе регистрируется своя копия текущего DVFS-регулятора. Поэтому в процессе работы DVFS-регулятора можно рассматривать только текущий обрабатываемый кластер.

Реализация DVFS-регуляторов для Android 11 находится в репозитории herolte_Eas_Idle_modification[13], для Android 10 в репозитории android_kernel_samsung_universal8890[14].

5.2.1. Алгоритм с двумя измерениями за 3 вызова регулятора

В первую очередь был реализован DVFS-регулятор, который максимально близок к алгоритму SPSA с двумя измерениями. Данный DVFS-регулятор называется spsa2_3 и требует три вызова для совершения одного полного цикла работы.

В процессе первого вызова:

- выясняется текущая частота — $curr\_i$;
- генерируется случайное число $\Delta = \pm 1$;
- выбирается частота — $new\_i = curr\_i + \Delta\beta$ (для первого зашумленного измерения алгоритма).

[13] https://github.com/makar-pelogeiko/herolte_Eas_Idle_modification/tree/Q-stable-spsa_gov

[14] https://github.com/makar-pelogeiko/android_kernel_samsung_universal8890/tree/lineage-17.1-spsa_gov

В процессе второго вызова:

- производится первое измерение — y^+, по средством вычисления функционала качества;
- выбирается частота — $new\_i = curr\_i - \Delta\beta$ (для второго зашумленного измерения алгоритма).

В процессе третьего вызова:

- производится второе измерение — y^-, по средством вычисления функционала качества;
- осуществляется проверка на корректность данных, чтобы успешно справляться с случаями, когда нагрузка резко возрастает или падает между измерениями;
- выбирается новая частота — $new\_i = curr\_i - \frac{\alpha(y^+ - y^-)}{2\Delta\beta}$.

5.2.2. Алгоритм с двумя измерениями за 2 вызова регулятора

Следующий DVFS-регулятор имеет название — spsa2_2. Он производит два измерения и требует два вызова для совершения одного полного цикла работы.

В процессе первого вызова:

- выясняется текущая частота — $curr\_i$;
- производится первое измерение — y^0, по средством вычисления функционала качества;
- генерируется случайное число $\Delta = \pm 1$;
- выбирается частота — $new\_i = curr\_i + \Delta\beta$ (для второго зашумленного измерения алгоритма).

В процессе второго вызова:

- производится второе измерение — y^+, по средством вычисления функционала качества;

- осуществляется проверка на корректность данных, чтобы успешно справляться с случаями, когда нагрузка резко возрастает или падает между измерениями;
- выбирается новая частота — $new\_i = curr\_i - \frac{\alpha(y^+ - y^0)}{\Delta\beta}$.

Такой подход позволяет сократить количество вызовов DVFS-регулятора, необходимых для одного полного цикла работы алгоритма, что должно позволить быстрее реагировать на изменение нагрузки на процессор.

5.2.3. Алгоритм с одним вызовом и двумя логическими измерениями

Для максимального сокращения количества необходимых вызовов для совершения одного полного цикла работы DVFS-регулятора был применен следующий подход.

При вызове DVFS-регулятора:

- выясняется текущая частота — $curr\_i$;
- генерируется случайное число $\Delta = \pm 1$;
- на основе текущей нагрузки моделируется нагрузка при частотах: $curr\_i \pm \Delta\beta$;
- на основе предположений из предыдущего пункта вычисляются функционалы качества: y^+ и y^-;
- осуществляется проверка корректности, так как данные о нагрузке получены путем моделирования;
- выбирается новая частота — $new\_i = curr\_i - \frac{\alpha(y^+ - y^-)}{2\Delta\beta}$.

Данный DVFS-регулятор имеет название spsa2_1_solid.

5.2.4. Модификация алгоритма для случаев высокой нагрузки на процессор

На основе алгоритма с одним вызовом и двумя логическими измерениями (spsa2_1_solid) было создано еще три алгоритма, один из которых получил название spsa2_1_highFreq. Поскольку нагрузка на кластер измеряется в процентах, то есть текущая нагрузка это число от 0 до 100 включительно, имеется сложность при нагрузке равной 100 процентам оценить, какую частоту следует выбрать. Для улучшения производительности при нагрузке 100% было решено увеличивать частоту больше чем требуется исходя из моделирования. При таком подходе может возникнуть ситуация, когда частота поднимается выше, чем необходимо. Однако нагрузка тоже может вырасти, а в противном случае неудачное решение будет исправлено следующим вызовом DVFS-регулятора.

5.2.5. Использование EAS для оценки нагрузки процессора

Была рассмотрена возможность оценивать нагрузку не стандартным для OnDemand способом, а при помощи EAS. для оценки производительности EAS использует понятие емкости ядра. Каждое ядро процессора имеет свою емкость (capacity) ($capacity \in \mathbb{Z}, capacity \in [0; 1024]$), которая зависит от ядра и его текущей частоты. Значения емкости ядер содержаться в энергетической модели (EM) и соответствуют производительности ядер на определенных частотах. Значение 1024 соответствует самому производительному ядру, работающему на максимальной доступной частоте, а остальные ядра и соответствующие им частоты имеют более низкие значения емкости. В это же время каждая задача тоже имеет свою емкость, то есть нагрузка на ядро процессора вычисляется как сумма емкостей всех задач в очереди на исполнение этого ядра. Процент загруженности ядра, используя EAS, можно получить как отношение используемой емкости ядра к текущей емкости этого ядра.

В свою очередь Schedutil вычисляет новую частоту следующим об-

разом: $next\_freq = C \cdot curr\_freq \cdot load$, где $next\_freq$ — следующая частота, $curr\_freq$ — текущая частота, C — константа, которая имеет значение 1.5 в нашем случае, $load$ — нагрузка на ядро в процентах.

Аналогичным образом была вычислена нагрузка в новом DVFS-регуляторе, который основан на spsa2_1_highFreq и имеет название — spsa2_1_EAS.

6. Сравнение существующих DVFS-регуляторов с разработанным

6.1. Android 11

6.1.1. Сравнение энергопотребления

На рисунках 2 — 7 отображены данные в мА·ч о энергопотреблении в соответствующих тестовых сценариях: *camera*, *flappyBird*, *trialXTreme3*, *twitch*, *type*, *videoVLC*. Каждый сценарий тестирования был выполнен 10 раз для каждого DVFS-регулятора. На графиках для каждого DVFS-регулятора отмечены: медианное, максимальное и минимальное значения. Для каждого DVFS-регулятора отображены данные полученные тремя способами, которые подробнее описаны в разделе 4.2. Синим цветом обозначен способ, не использующий данные о времени простоя процессора, оранжевым — способ, при котором происходит вычитание времени простоя из минимальной частоты, и зеленым цветом оторажен способ, при котором время простоя вычитается пропорционально времени проведенному на определенной частоте. После названия регулятора указаны использованные параметры, если таковые были.

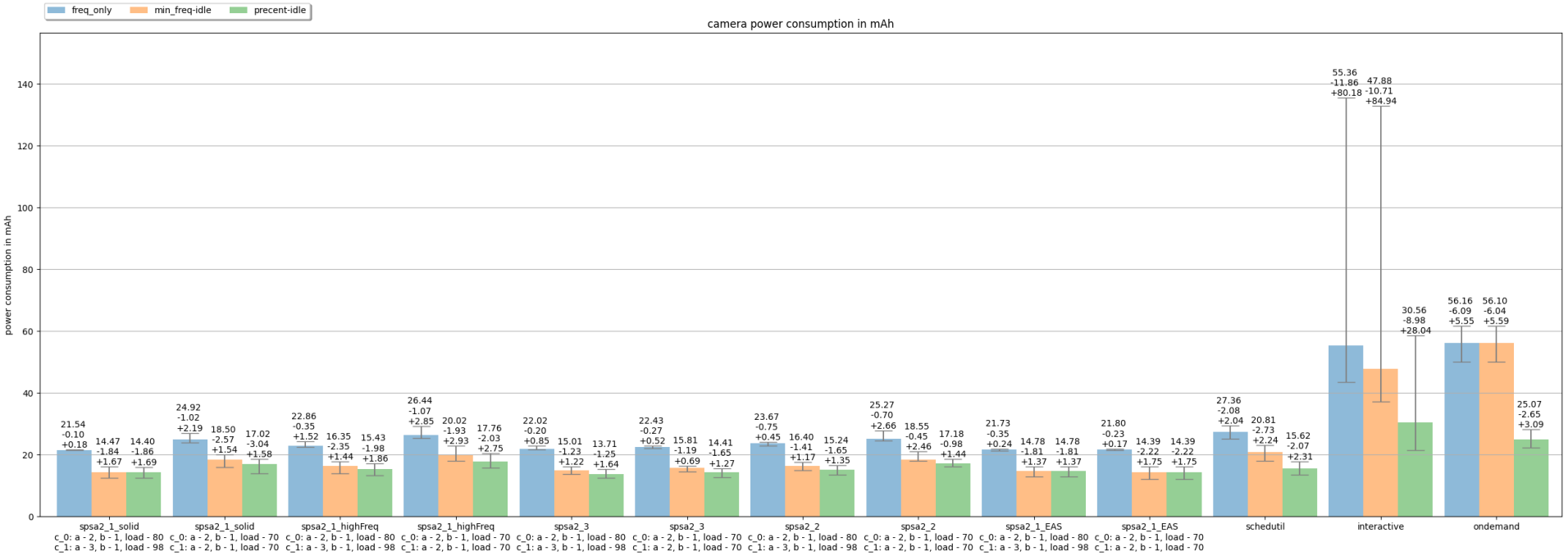

Рис. 2: График энергопотребления в тесте camera для Android 11.

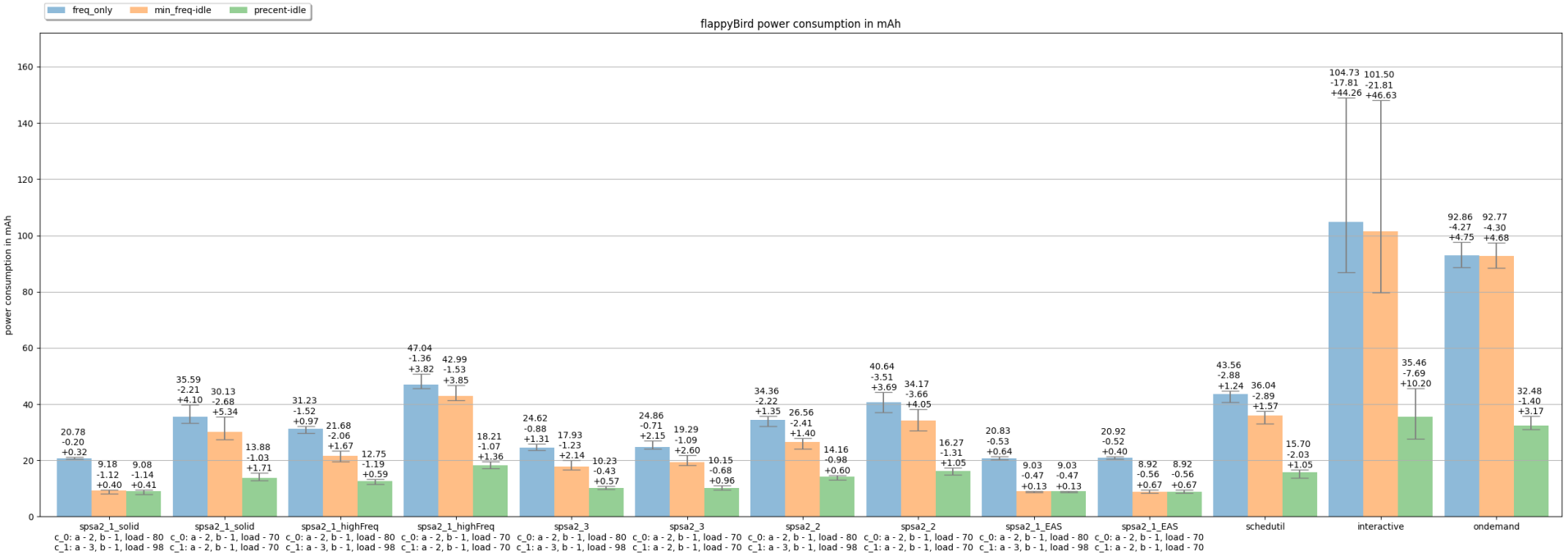

Рис. 3: График энергопотребления в тесте flappyBird для Android 11.

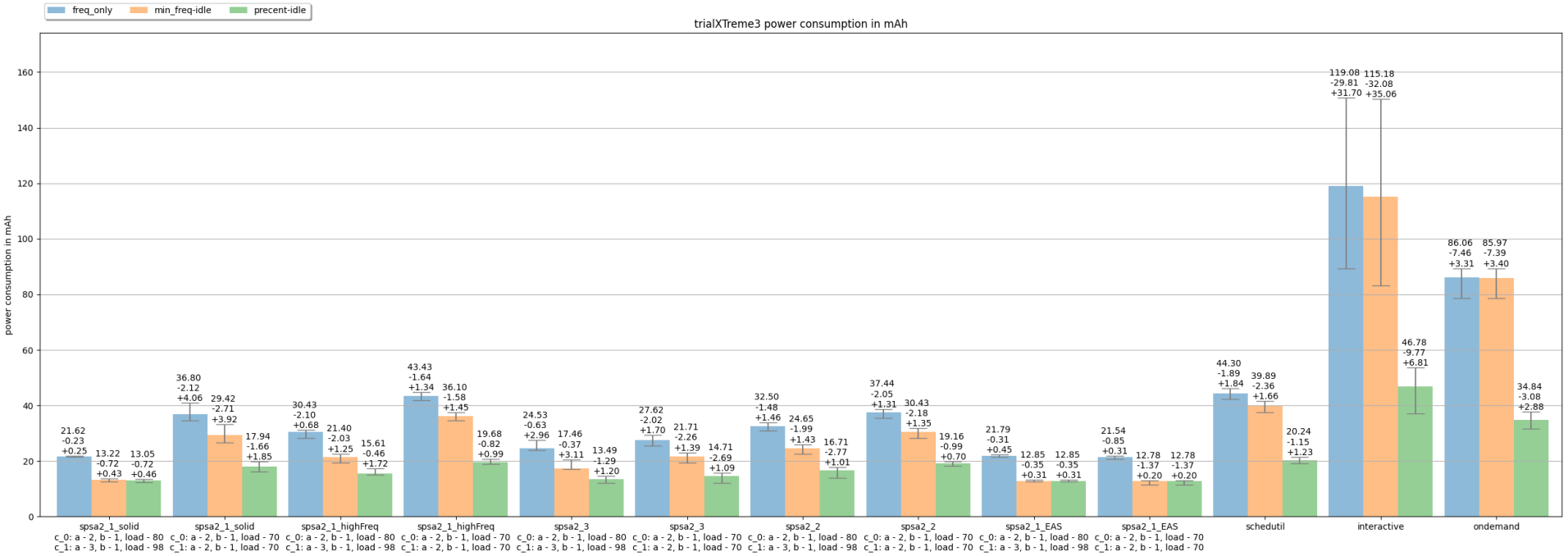

Рис. 4: График энергопотребления в тесте trialXTreme3 для Android 11.

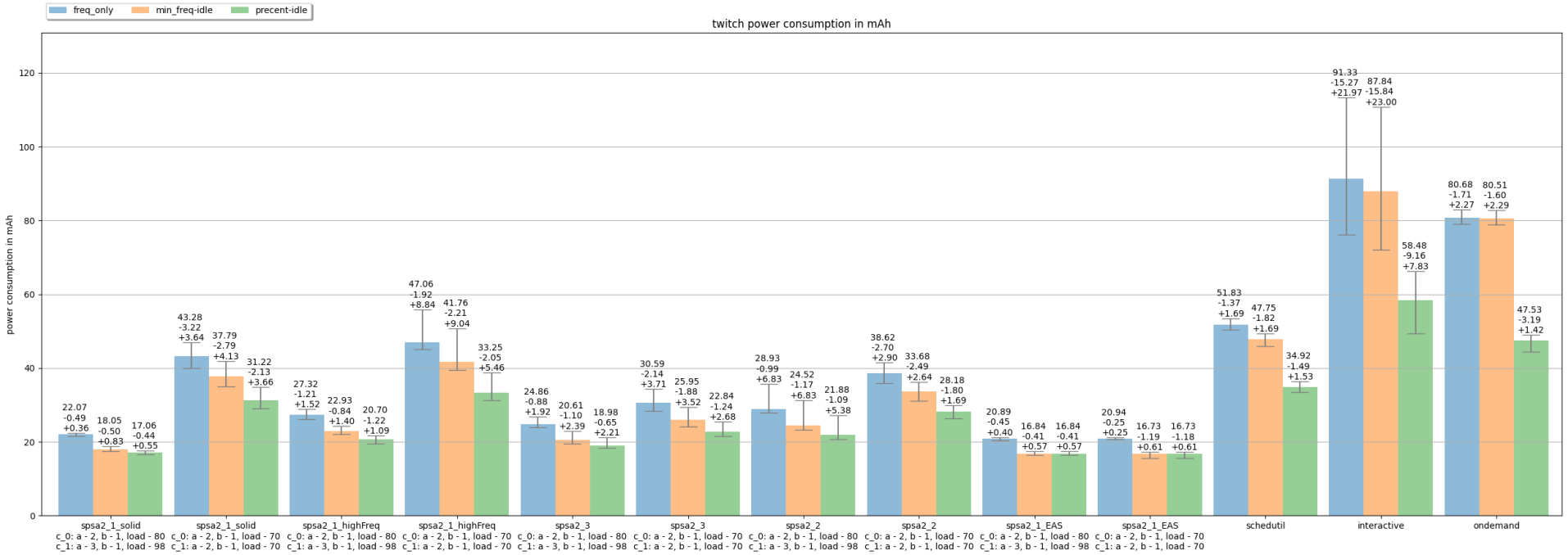

Рис. 5: График энергопотребления в тесте twitch для Android 11.

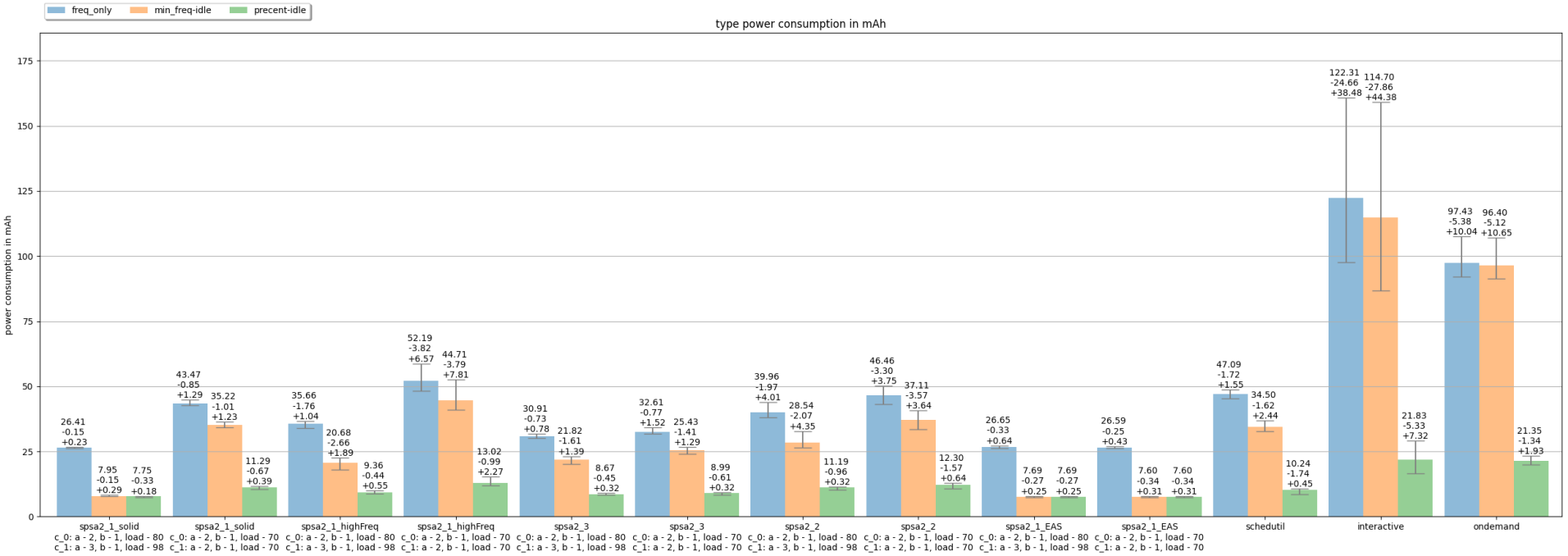

Рис. 6: График энергопотребления в тесте type для Android 11.

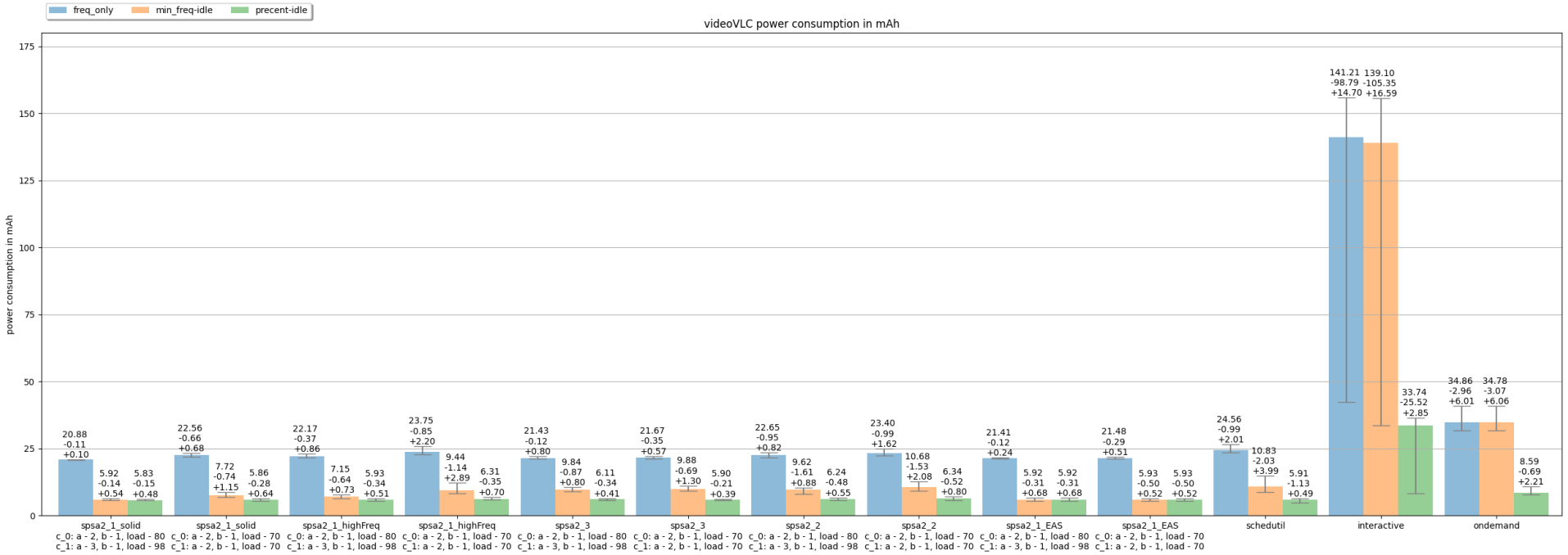

Рис. 7: График энергопотребления в тесте videoVLC для Android 11.

6.1.2. Сравнение производительности

В таблице 1 представлены медианные значения результатов запуска бенчмарка Geekbench 5.5.1 [11]. Для каждого регулятора было выполнено 3 запуска бенчмарка. Для обобщения данных были вычислены средние арифметические значения. Более подробные результаты находятся в таблице
DVFS_govs_performance_comparison.xlsx[15].

GeekBench 5.5.1: PERFORMANCE, lineage 18.1_bf-16, score points									
SPSA tuners		spsa2_1_solid	spsa2_1_highFreq	spsa2_3	spsa2_2	spsa2_1_EAS	schedutil	interactive	ondemand
α: 2, β: 1, load: 70	single core	319	331	287	302	83	319	321	296
	multicore	1060	1073	970	1025	281	970	1112	1068
cluster 0: α: 2, β: 1, load: 80; cluster 1: α: 3, β: 1, load: 98	single core	83	324	282	290	83			
	multicore	564	966	913	962	275			

Таблица 1: Результаты замеров производительности в score points Geekbench 5.

6.2. Android 10

6.2.1. Сравнение энергопотребления

Для сравнения DVFS-регуляторов без использования EAS использовался Android 10. Для сравнения были взяты все DVFS-регуляторы из предыдущего тестирования за исключением регуляторов, которые не работают без EAS: Schedutil и spsa2_1_EAS. Тестирование и построение графиков происходило аналогично предыдущему тестированию.

Общее измерение производительности не производилось, и более детальный анализ будет проведён далее в тексте. На рисунках 8 — 13 отображены данные в мА·ч о энергопотреблении в соответствующих тестовых сценариях: *camera*, *flappyBird*, *trialXTreme3*, *twitch*, *type*, *videoVLC*.

[15] https://studentspburu-my.sharepoint.com/:f:/g/personal/st076963_student_spbu_ru/EkZ5ufG6tRZIuZKezfPGlr0BQLc4cphDILYbUvdT1_CIMg?e=jdc2UA

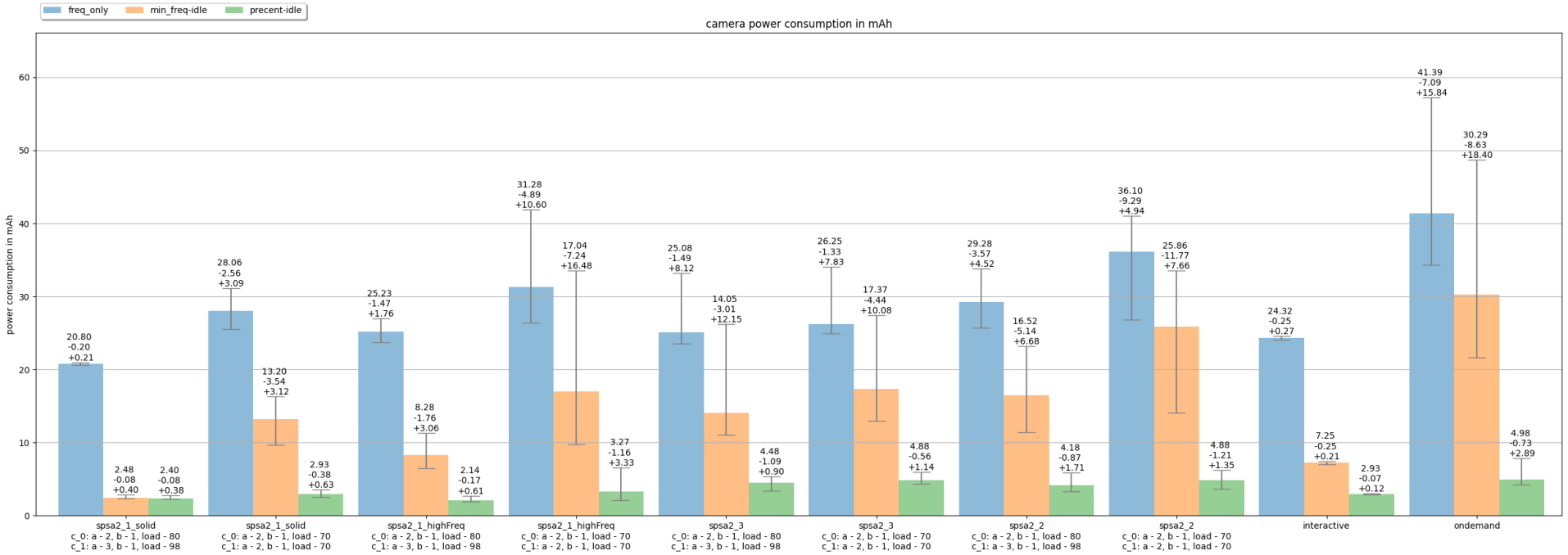

Рис. 8: График энергопотребления в тесте camera для Android 10.

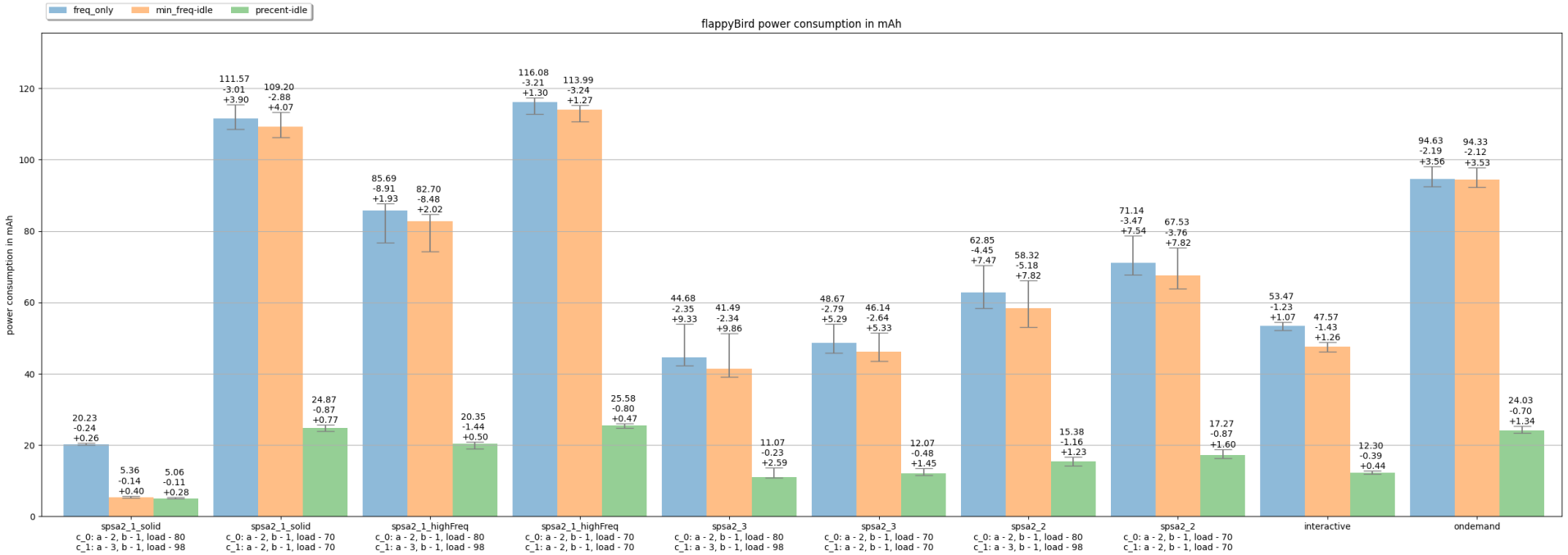

Рис. 9: График энергопотребления в тесте flappyBird для Android 10.

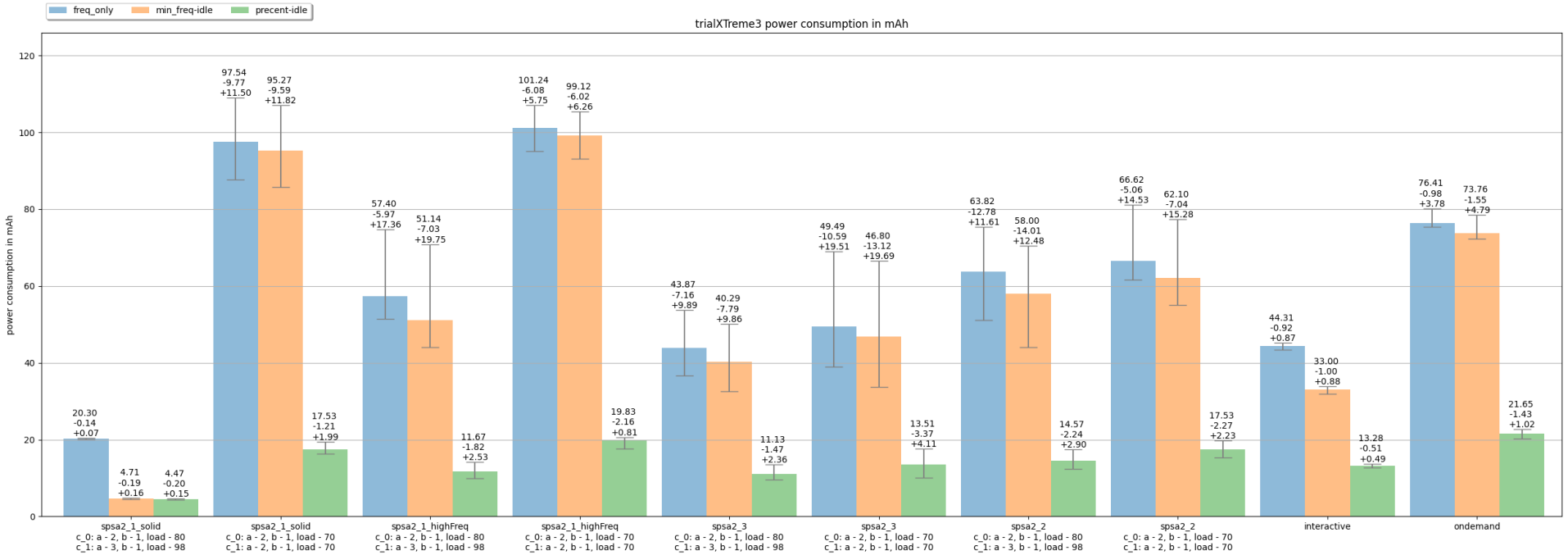

Рис. 10: График энергопотребления в тесте trialXTreme3 для Android 10.

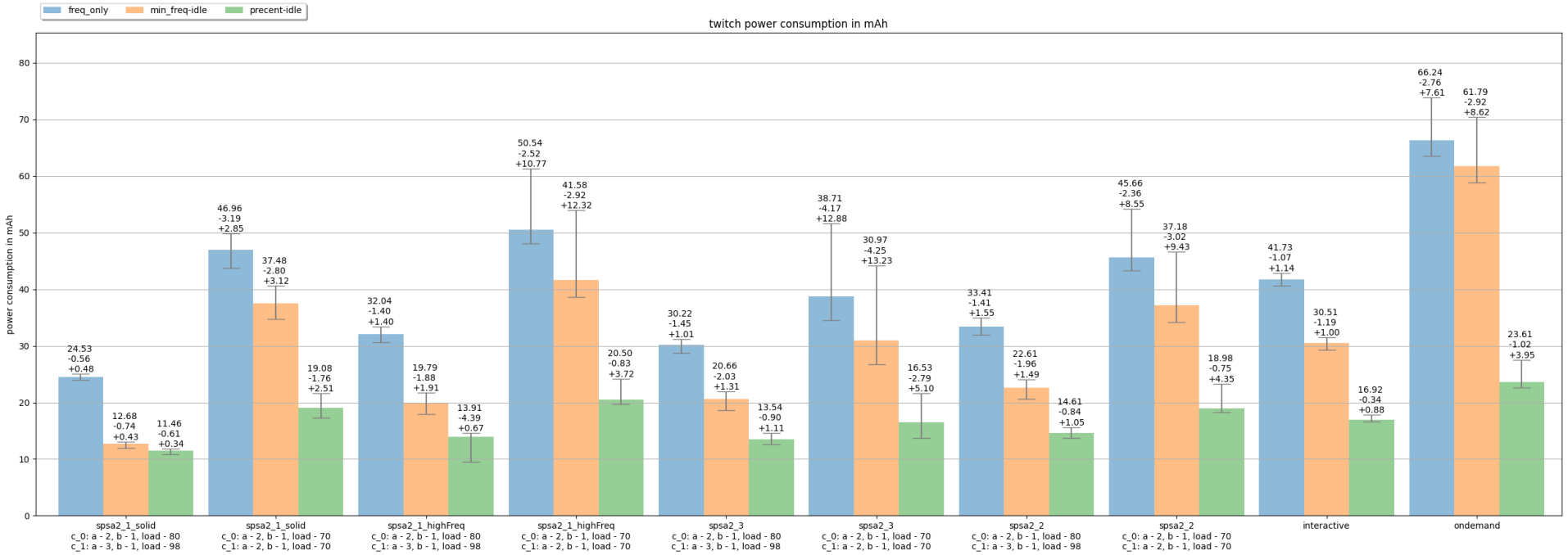

Рис. 11: График энергопотребления в тесте twitch для Android 10.

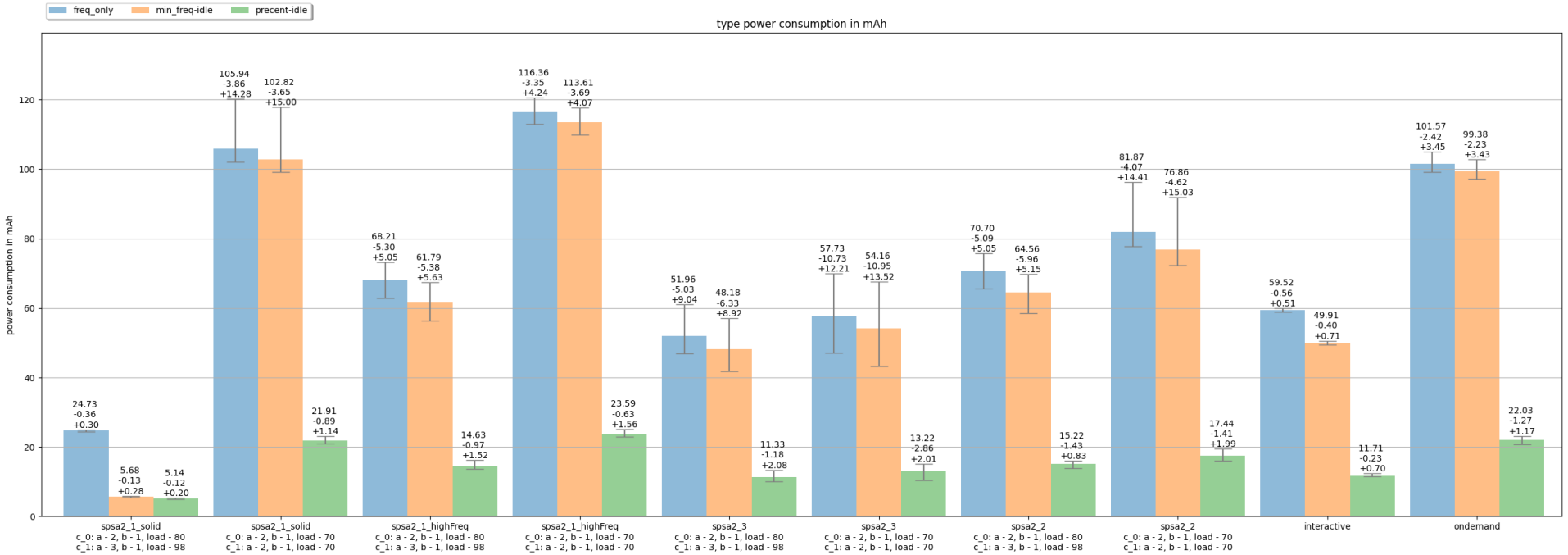

Рис. 12: График энергопотребления в тесте type для Android 10.

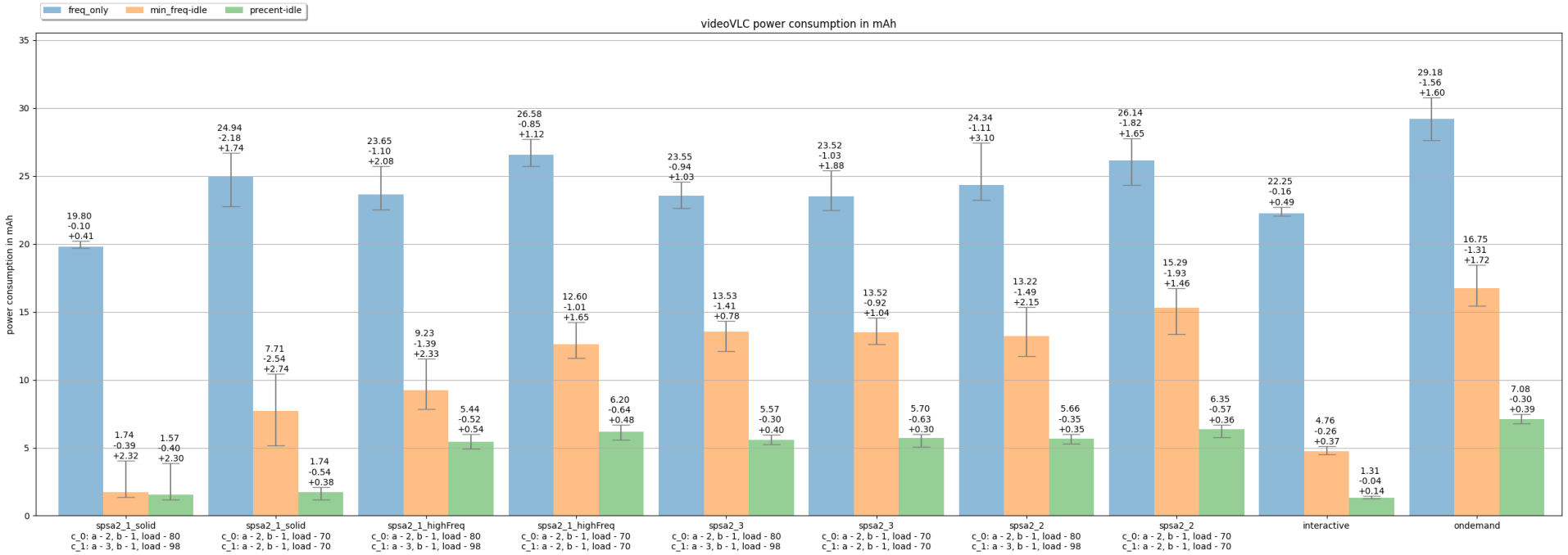

Рис. 13: График энергопотребления в тесте videoVLC для Android 10.

6.3. Анализ результатов

Исходя из полученных результатов, видно, что DVFS-регулятор spsa2_1_EAS имеет самое низкое энергопотребление среди всех сравниваемых DVFS-регуляторов. Однако данный DVFS-регулятор имеет самую низкую производительность. Такой результат связан с тем, что примененный способ оценки нагрузки на процессор плохо подходит для использования в данном алгоритме. В большинстве случаев нагрузка недооценивалась, что приводило к выбору более низких частот, из-за чего получилось низкое энергопотребление и низкая производительность.

При сравнении энергопотребления Schedutil, Interactive и OnDemand лучшие результаты показал именно Schedutil.

Как показано в таблице 1 DVFS-регуляторы spsa2_1_solid и spsa2_1_highFreq с параметрами:

- кластер 0:
  - $\alpha = 2$,
  - $\beta = 1$,
  - target_load = 80;
- кластер 1:
  - $\alpha = 3$,
  - $\beta = 1$,
  - target_load = 98.

показали сильно отличающуюся производительность: spsa2_1_solid имеет отставание более чем в 11 раз относительно Schedutil, а spsa2_1_highFreq показал сопоставимую производительность со Schedutil. В это же время энергопотребление оказалось ниже как в сравнении со Schedutil
(spsa2_1_solid показал сокращение примерно на 14% — 57%,

spsa2_1_highFreq — на 9% — 31%), так и в сравнении с этими же DVFS-регуляторами, использующими параметры:

- $\alpha = 2$,
- $\beta = 1$,
- target_load = 70.

для обоих кластеров. Однако использование данных параметров привело к более высоким результатам производительности, которые могут даже превосходить производительность Schedutil в некоторых случаях. Исходя из результатов сравнения энергопотребления, видно, что данные DVFS-регуляторы показали близкие к Schedutil значения:
spsa2_1_solid показал сокращение энергопотребления на 8% — 16%; spsa2_1_highFreq показал энергопотребление больше, чем Schedutil, в сценариях тестирования *type*, *flappyBird* на 11% и 8% соответственно, и меньше на 9% в сценарии тестирования *twitch*, а в остальных сценариях тестирования spsa2_1_highFreq показал сокращение энергопотребления на 1% — 3%.

DVFS-регуляторы spsa2_3 и spsa2_2 с параметрами:

- $\alpha = 2$,
- $\beta = 1$,
- target_load = 70.

показали сокращение энергопотребления на 12% — 43% и 1% — 25% относительно Schedutil. Производительность при многоядерной нагрузке практически не изменилась, а при нагрузке на одно ядро для spsa2_3 стала меньше на 10%, для spsa2_2 стала меньше на 5%. При тестировании DVFS-регуляторов spsa2_3 и spsa2_2 с параметрами:

- кластер 0:
  - $\alpha = 2$,
  - $\beta = 1$,

  - target_load = 80;

- кластер 1:

  - $\alpha = 3$,
  - $\beta = 1$,
  - target_load = 98.

энергопотребление стало ниже на 12% — 52% и 7% — 44% относительно Schedutil. Производительность при нагрузке на одно ядро оказалась меньше чем у Schedutil на 11% и 9%, а при многоядерной нагрузке — для spsa2_3 меньше на 5%, а для spsa2_2 меньше на 1% . При сравнении данных DVFS-регуляторов между собой видно, что производительность выше у spsa2_2, а энергопотребление ниже у spsa2_3. Это связано с необходимым количеством вызовов регулятора для совершения одного полного цикла работы алгоритма.

После проведения сравнения DVFS-регуляторов без EAS с использованием Android 10 было выяснено, что при нагрузке 99% — 100% данный алгоритм не сразу рекомендует увеличить частоту, из-за чего очень сильно снизилась производительность. Остальные, разработанные, DVFS-регуляторы показали энергопотребление, уступающее DVFS-регулятору Interactive. Поэтому общее сравнение производительности не производилось.

Причиной увеличения энергопотребления новых DVFS-регуляторов без EAS стало распределение задач между ядрами. При использовании EAS LITTLE-ядра нагружаются больше, чем big, если нагрузка на процессор меньше 80%, в то время как при использовании стандартного планировщика (completely fair scheduler, CFS) нагрузка распределялась равномерно между всеми ядрами, из-за чего big ядра нагружались больше в сравнении с аналогичными ситуациями при использовании EAS.

Заключение

В ходе работы были достигнуты следующие результаты.
Проведен обзор существующих DVFS-регуляторов:

- изучены стандартные регуляторы - Schedutil, OnDemand, Interactive;
- рассмотрены подходы к созданию новых регуляторов: добавление возможности перевода ядер в сон, добавление анализа характера выполняемых инструкций, использование стохастической аппроксимации.

Подготовлен тестовый стенд:

- выбран Samsung SM-G930F;
- подобрано, установлено и настроено соответствующее ПО для проведения дальнейшей работы.

Создана методика тестирования энергопотребления смартфона:

- выбраны способ взаимодействия со смартфоном, способ оценки энергопотребления, сценарии тестирования;
- создан комплекс утилит для тестирования энергопотребления процессора смартфона.

Разработан регулятор частоты на основе алгоритма SPSA с двумя измерениями. В процессе разработки было создано несколько алгоритмов: spsa2_1_solid, spsa2_1_highFreq, spsa2_2, spsa2_3, spsa2_1_EAS, которые отличаются друг от друга количеством необходимых вызовов для одного шага алгоритма, способом измерения, алгоритмом выбора следующей оптимальной оценки.

Проведено сравнение разработанных DVFS-регуляторов с существующими Schedutil, Interactive, OnDemand. Исходя из полученных результатов, алгоритм spsa2_3 показал меньшее энергопотребление (на 12% — 52%) и малое отставание производительности (на 5% — 11%)

относительно schedutil, а алгоритм spsa2_2 более высокую производительность и более высокое энергопотребление относительно spsa2_3. Алгоритм spsa2_1_solid и spsa2_1_highFreq показали энергопотребление и производительность ниже, чем schedutil. Производительность spsa2_1_highFreq выше, чем у остальных разработанных DVFS-регуляторов, в то время как производительность spsa2_1_solid в 11 раз ниже относительно schedutil при использовании высокой целевой нагрузки. При повышении целевой нагрузки у всех разработанных DVFS-регуляторов снижается производительность и энергопотребление.

Список литературы

[1] ARM Ltd. Arm Big.LITTLE. — URL: https://www.arm.com/technologies/big-little (дата обращения: 2023-04-25).

[2] ARM Ltd. EAS Overview and Integration Guide. — 2018. — URL: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Open%20Source/energy-aware-scheduling/eas_overview_and_integration_guide_r1p6.pdf (дата обращения: 2023-04-27).

[3] Frumusanu A. Exynos 8890. — URL: https://www.anandtech.com/show/10075/early-exynos-8890-impressions (дата обращения: 2023-04-10).

[4] Google LLC. Supporting multiple CPU idle levels in kernel. — URL: https://android.googlesource.com/kernel/msm/+/android-msm-marlin-3.18-nougat-dr1/Documentation/cpuidle/sysfs.txt (дата обращения: 2023-04-14).

[5] Google LLC. Android debug bridge. — 2023. — URL: https://developer.android.com/tools/adb (дата обращения: 2023-04-09).

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

[7] MobiCore: An adaptive hybrid approach for power-efficient CPU management on Android devices / Lucie Broyde, Kent Nixon, Xiang Chen et al. // 2017 30th IEEE International System-on-Chip Conference (SOCC). — 2017. — P. 221–226. — URL: http://doi.acm.org/10.1145/1159876.1159880.

[8] Ohk Seung-Ryeol, Kim YongSin, Kim Young-Jin. Phase-Based Low Power Management Combining CPU and GPU for Android Smartphones // Electronics. — 2022. — Vol. 11, no. 16. — URL: https://www.mdpi.com/2079-9292/11/16/2480.

[9] On Application of Simultaneous Perturbation Stochastic Approximation for Dynamic Voltage-Frequency Scaling in Android OS / Evgenii Bogdanov, Alexander Bozhnyuk, Stanislav Sartasov, Oleg Granichin // 2021 7th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP). — 2021. — P. 1–7.

[10] Pallipadi V. Linux cpufreq - stats driver. — URL: https://android.googlesource.com/kernel/common/+/a7827a2a60218b25f222b54f77ed38f57aebe08b/Documentation/cpu-freq/cpufreq-stats.txt (дата обращения: 2023-04-14).

[11] Primate Labs Inc. Geekbench 5 CPU Workloads. — URL: https://www.geekbench.com/doc/geekbench5-cpu-workloads.pdf (дата обращения: 2023-04-14).

[12] Samsung Electronics Co Ltd. Benefits of the big.LITTLE Architecture. — URL: https://s3.ap-northeast-2.amazonaws.com/global.semi.static/Benefits_of_the_bigLITTLE_Architecture.pdf (дата обращения: 2023-04-25).

[13] Samsung Electronics Co Ltd. SM-G930F. — URL: https://www.samsung.com/levant/support/model/SM-G930FZSASEE (дата обращения: 2023-04-10).

[14] Snowdon David, Ruocco Sergio, Heiser Gernot. Power management and dynamic voltage scaling: Myths and facts. — 2005. — 01.

[15] The community kernel development?? Energy Aware Scheduling. — URL: https://docs.kernel.org/scheduler/sched-energy.html (дата обращения: 2023-04-10).

[16] Wysocki R. J. CPU Performance Scaling. — URL: https://www.kernel.org/doc/html/v4.14/admin-guide/pm/cpufreq.html (дата обращения: 2023-04-10).

[17] А. Т. Вахитов О. Н. Граничин. Рандомизированные алгоритмы оценивания и оптимизации при почти произвольных помехах. — Наука, 2003. — P. 291. — ISBN: 5020065250, 9785020065253.