Санкт-Петербургский государственный университет

Программная инженерия
кафедра системного программирования

Евгений Алексеевич Богданов

Использование стохастической оптимизации для регулировки частоты процессора в Android OS

Выпускная квалификационная работа

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

Консультант:
ст. преп. Сартасов С. Ю.

Рецензент:
к.ф.-м.н. Иванский Ю.В.

Санкт-Петербург
2021

SAINT-PETERSBURG STATE UNIVERSITY

Software engineering

Evgenii Bogdanov

Using Stochastic Optimization for CPU Frequency regulation in Android OS

Bachelor's Thesis

Scientific supervisor:
Doctor of Science, Professor Granichin O. N.

Adviser:
Senior Lecturer Sartasov S. Y.

Reviewer:
candidate Ivanskiy Y. V.

Saint-Petersburg
2021

Оглавление

Введение **5**

1. Цели и задачи **7**

2. Существующие подходы в области оптимизации энергопотребления **8**

2.1. Регуляторы частоты ОС Android 8

2.2. Альтернативные подходы к оптимизации 12

2.2.1. Алгоритм поиска статей 12

2.2.2. Адаптивные алгоритмы 13

2.2.3. Машинное обучение 15

2.2.4. Предварительные вычисления 19

2.2.5. Выводы 21

3. Инфраструктура для загрузки алгоритмов DVFS **22**

3.1. Xiaomi Redmi Note 8 Pro 22

3.2. Права суперпользователя 22

3.3. Особенности работы со сторонними ядрами и прошивками 23

4. Предлагаемый алгоритм DVFS **25**

4.1. Стохастическая аппроксимация со случайными направлениями 25

4.2. Модель состояния системы 27

4.3. Алгоритм DVFS 28

5. Особенности реализации алгоритма **29**

5.1. Поддержка многоядерности 29

5.2. Поддерживаемые настройки 30

6. Тестирование **31**

6.1. Инструменты и критерии тестирования 31

6.2. Методология тестирования 32

6.3. Анализ результатов 35

Заключение **37**

Список литературы **38**

Введение

Мобильные устройства сегодня являются неотъемлемой частью жизни современного человека. Важной задачей является увеличение длительности автономной работы этих устройств, т.е. чтобы они работали как можно дольше от аккумулятора, без дополнительной подзарядки. Вместе с тем с каждым годом растет мощность мобильных устройств: появляются новые процессоры со всё большим числом ядер и со всё более сложными вычислительными модулями, которые потребляют значительную часть энергии для своей работы, что сильно снижает время автономной работы мобильных устройств.

Однако большинство прикладных задач, например, отрисовка меню, показ несложных анимаций, обработка очередного видео-кадра и т.д. не требуют таких мощностей. Поэтому очень важно научиться находить баланс между производительностью процессора и энергией, которую он реально потребляет для своей работы в настоящий момент.

Существует ряд исследований, которые показали, что энергопотребление процессора определяется следующим законом [7]: $P \sim fu^2$, где P — мощность, потребляемая процессором, f — его частота и u — напряжение. Следовательно, меняя два этих параметра можно достичь искомого баланса между производительностью и энергопотреблением.

С этой целью в ОС Android существует подсистема CPUFreq subsystem [20], которая предоставляет интерфейс для разработки сторонних алгоритмов по регулировке частоты и напряжения процессора. Эти алгоритмы принято называть алгоритмами DVFS (Dynamic Voltage and Frequency Scaling) или, что тоже самое, DVFS-регуляторами.

Таким образом, реализовав сторонний алгоритм DVFS , можно динамически определять оптимальное P-состояние (конфигурацию частоты и напряжения), в котором должен находиться процессор, для достижения наилучшего сочетания его производительности и энергоэффективности.

В операционной системе Android существует несколько реализаций подобных алгоритмов, однако, используемые в них подходы не опти-

мальны. Особенно это проявляется при работе с современными многоядерными архитектурами. Также у некоторых из этих решений присутствуют и алгоритмические недостатки, например, частые смены сильно отличающихся друг от друга P-состояний, что приводит к существенным потерям энергии при переключении между ними и сильно снижает эффективность подобного подхода [35].

В связи с этим на протяжении последних нескольких лет проводится большое количество исследований более новых и современных подходов к разработке различных алгоритмов DVFS. Все они подразделяются на 3 большие группы: адаптивные алгоритмы [18, 15], машинное обучение [22, 3, 31] и алгоритмы, использующие предварительные вычисления для своей работы [16, 32].

Однако в данный момент не существует алгоритмов регулировки частот, которые базируются на использовании методов стохастической оптимизации. В тоже время подобные методы широко используются для решения схожих оптимизационных задач. Данная работа призвана восполнить этот пробел. Планируется разработать алгоритм регулировки частот процессора на базе стохастической аппроксимации со случайными направлениями. (Simultaneous perturbation stochastic approximation — SPSA [38, 17]).

1. Цели и задачи

Цель данной дипломной работы является разработка алгоритма DVFS с использованием стохастической оптимизации и его апробация на реальном мобильном устройстве.

Для достижения этой цели были сформулированы следующие задачи.

1. Провести обзор существующих подходов в области оптимизации энергопотребления.

2. Выбрать смартфон и создать инфраструктуру для загрузки алгоритмов DVFS.

3. Разработать новый алгоритм DVFS и реализовать его для ОС Android.

4. Сравнить полученный алгоритм с уже существующими алгоритмами DVFS.

2. Существующие подходы в области оптимизации энергопотребления

2.1. Регуляторы частоты ОС Android

В этой главе представлен обзор регуляторов частоты, использующихся в ОС Android [35, 20, 21] по умолчанию, начиная от самых простых, держащих частоту на одном уровне, и заканчивая более сложными, меняющими частоту в зависимости от нагрузки на процессор или пользовательского взаимодействия с устройством.

Performance — держит частоту на максимально возможном уровне, что позволяет достичь максимальной производительности устройства.

PowerSave — держит частоту на минимально возможном уровне и позволяет существенно увеличить время работы устройства от аккумулятора.

UserSpace — позволяет пользователю или любой программе с root-правами устанавливать свою частоту ядра процессора.

Ondemand — один из когда-то самых популярных и эффективных регуляторов, на базе которого создано множество более современных подходов к управлению частотами процессора. Для своей работы он использует нагрузку на процессор в качестве показателя, по которому выбирает целевую частоту процессора.

Нагрузка на процессор вычисляется следующим образом: между вызовами своей рабочей процедуры регулятор измеряет время, на протяжении которого процессор был в активном состоянии, а затем вычисляет отношение этого времени к общему времени между вызовом рабочей процедуры.

Частота выбирается пропорционально этой нагрузке, однако если нагрузка превышает порог, указанный в параметре up_threshold, то целевой частотой становится самая высокая частота, которую можно установить в системе.

Один из главных недостатков этого регулятора заключается в рез-

ких скачках частоты, с которой работает процессор.

Параметры регулятора:

- sampling_rate (в микросекундах — 10^{-6} сек): указывает, как часто нужно проверять нагрузку на процессор. Его значение по умолчанию равно transition_latency умножить на 1000, где transition_latency — параметр, находящийся в в файле cpuinfo_transition_latency в подкаталоге cpufreq, измеряемый в наносекундах (10^{-9} сек) и показывающий, сколько времени будет тратить процессор при переключении с одной частоты на другую (равен -1, если параметр неизвестен) * 1000, потому что задержка измеряется в наносекундах. Корректируется в зависимости от transition_latency;

- sampling_rate_min: ограничение на частоту проверок снизу. Изначально ограничение равно transition_latency * 100 или зависит от ограничений ядра;

- up_threshold (в %): если нагрузка на процессор превышает это значение, то регулятор ставит частоту на максимальное значение в рамках политики, иначе частота выставляется пропорционально нагрузке;

- ignore_nice_load (0 или 1): по умолчанию установлено в 0. В этом случае при расчете нагрузки учитываются все процессы, подсчет ведется по параметру — загрузка процессора. Если установить 1, то процессы, выполняемые со значением «nice», учитываться не будут при расчете общей нагрузки. Иначе говоря, если есть процессы, которые не нужно учитывать при подсчете, можно установить им уровень «nice» выше нуля и установить текущий параметр в 1;

- sampling_down_factor (1..100): это множитель, который применяется к sampling_rate, если нагрузка на процессор превышает up_threshold. При установке в 1 (по умолчанию) оценка ча-

стоты происходит независимо от частоты с одинаковым интервалом. Больше единицы устанавливается при нагрузке выше up_threshold. Это позволяет процессору дольше оставаться с фиксированной максимальной частотой. Также снижаются накладные расходы на оценку нагрузки.

Conservative — регулятор, также как и Ondemand, работающий в зависимости от текущей нагрузки на процессор, однако позволяющий избежать значительного изменения частоты за короткие промежутки времени. Для этого он меняет частоту небольшими шагами, указанных в параметрах freq_step и sampling_down_factor, в зависимости от того, превышено пороговое (параметр down_threshold) значение или нет.

Параметры регулятора:

- down_threshold (в %): пороговое значение (по умолчанию 20%), определяющее, как будет меняться частота: если нагрузка больше этого значения то частота будет увеличена на значение параметра freq_step, если меньше то частота будет уменьшена в зависимости от sampling_down_factor;

- freq_step (в %): процент от максимально доступной частоты (по умолчанию 5%) на который будет изменяться текущая частота системы. Если указан 0, то будет использовано значение по умолчанию, если указано 100, то регулятор будет переключаться между максимальной и минимальной частотой в системе;

- sampling_down_factor: коэффициент уменьшения убывания частоты от 1 до 10. При установке этого коэффициента частота будет понижаться в sampling_down_factor раз медленнее, чем повышаться.

Interactive — регулятор, разработанный специально для устройств, чувствительных к задержкам при работе (например, при отрисовке пользовательского интерфейса).

Одним из недостатков регулятора onDemand является возможность изменения частоты только с определенной периодичностью. Это может привести к тому, что между этими изменениями частота процессора будет не оптимальной, например, слишком низкой, что не позволит быстро отрисовать какой-нибудь пользовательский интерфейс. Эту проблему решает Interactive. Он меняет частоту процессора не только с определенной периодичностью, но и в момент выхода из простоя.

При выходе из простоя на протяжении задержки, указанной в параметре min_sample_time измеряется нагрузка на процессор, а затем анализируется это значение:

1. Если нагрузка превысила порог go_hispeed_load, то частота повышается до максимальной.

2. Если текущая нагрузка недостаточно велика (меньше go_hispeed_load), то она сравнивается с нагрузкой в момент последней регулировки частоты, а затем частота выставляется пропорционально максимуму среди этих двух величин.

Параметры регулятора:

- min_sample_time (в мкс): сколько времени провести на текущей частоте перед тем, как снизить частоты. Нужно для подсчета нагрузки с момента регулировки частоты;

- go_hispeed_load (в %): нагрузка процессора, при которой происходит переход к высокой частоте (по умолчанию 85%);

- hispeed_freq: "промежуточная высокая скорость" — частота, на которую переключается процессор при превышении порога go_hispeed_load. Если нагрузка сохранятся в течении above_hispeed_delay, то скорость может быть дополнительно увеличена (по умолчанию максимальная частота в системе);

- above_hispeed_delay (в мкс): как только частота достигла go_hispeed_load, подождать это время, прежде чем поднимать

частоту еще выше в ответ на продолжающуюся высокую нагрузку (по умолчанию — 20000 мкс);

- timer_rate: частота, с которой проверяется нагрузка на процессор вне режима простоя;
- input_boost: изначально равен 0. Если не 0, увеличить частоту всех ядер до hispeed_freq при работе с экраном;
- boost: изначально равен 0. Если не 0, увеличить частоту всех ядер до hispeed_freq, пока снова не будет записано 0;
- boostpulse: если не 0, то немедленно поднять частоту всех ядер до hispeed_freq на время min_sample_time, а затем действовать как обычно.

2.2. Альтернативные подходы к оптимизации

В рамках данной работы также были изучены и альтернативные, более прогрессивные, подходы к оптимизации энергопотребления, которые существенно выигрывают у базовых регуляторов, описанных в главе 2.1. Для этого были исследованы работы по классификации таких алгоритмов уже проводившиеся на математико-механическом факультете СПБГУ, изучены статьи, лежащие в основе этих работ, а также найдены новые, наиболее актуальные подходы к оптимизации энергопотребления на данный момент.

В следующих подглавах представлен алгоритм поиска статей для исследования, а затем их описание сгруппированное по трем основным типам используемых подходов.

2.2.1. Алгоритм поиска статей

Так как исследуемая область активно развивается и каждый год появляется большое количество новых работ, так или иначе затрагивающих тему энергопотребления в мобильных устройствах, необходимо выбрать наиболее актуальные статьи для данной дипломной рабо-

ты. Для этого автором работы совместно со студентом математико-механического факультета СПбГУ — Александром Божнюком, проходящим учебную практику в той же области, был воспроизведен следующий алгоритм поиска статей:

1. Сформулированы основные вопросы, ответы на которые должна содержать изучаемая работа.

2. Составлены запросы к поисковой системе "Google Scholar" содержащие ключевые слова, которые по мнению авторов должна содержать статья:

   - DVS Dynamic Voltage Scaling power consumption algorithms OS android;
   - DVFS algorithm android.

3. Изучены первые 2 страницы выдачи поисковой системы (20 статей) в соответствии с одним из запросов и отобраны статьи, которые по мнению автора отвечают на поставленные вопросы и подлежат изучению.

4. Далее авторы обменялись отобранными работами и проверили их соответствие сформулированным требованиям. Таким образом выбранные статьи прошли двойной отбор и были утверждены каждым из участников.

Результатом работы на данном шаге стали 6 статей, к которым были добавлены ещё 10 статей лежащих в источниках работ [30], проведенных ранее на математико-механическом факультете. Полученные статьи были поделены между авторами и изучены более детально, ниже представлен обзор статей изученных непосредственно автором данной дипломной работы.

2.2.2. Адаптивные алгоритмы

В статье [18] авторы описывают собственный DVFS регулятор, который построен на базе классического onDemand, дополненного воз-

можностью включать и отключать неиспользуемые ядра (переводить в состояние простоя), а также показывают способ пересчета частот при отключенных ядрах.

Основная идея подхода заключается в следующем:

1. Выбран порог для отключения ядра процессора равный 10%. Если нагрузка на ядро меньше порога, то это ядро переходит в состояние простоя.

2. Представлена формула пересчета частот процессора на базе работы регулятора onDemand:

$$n \cdot f_{new} = n_{max} \cdot f_{ondemand} \cdot K$$

   где n — количество активных ядер, n_{max} — общее количество ядер, f_{new} — новая частота ядра, $f_{ondemand}$ — результат работы регулятора ondemand, K — общая нагрузка на телефон.

3. Сформулирован алгоритм регулировки пропускной способности ядра:

   — подсчитываем изменение загруженности ядра в моменты времени t и $t-1$;

   — если это изменение выше порога upTherehold, то увеличиваем пропускную способность ядра;

   — если это изменение меньше порога downTherehold, то уменьшаем пропускную способность ядра.

4. Показан общий алгоритм работы системы:

   — запускаем регулятор ondemand;

   — для каждого ядра определяем нужно ли изменить пропускную способность. Если да, то меняем;

   — определяем нужно ли отключить какие-либо ядра;

   — отключаем ядра и обновляем частоты.

Замеры, проведенные авторами, показывают, что, в сравнении с базовыми регуляторами в Android, данный подход позволяет экономить до 14% энергии для динамических нагрузок на смартфон и до 22% при более постоянной нагрузке.

В работе [15] представлено другое решение, базирующееся на использовании базовых регуляторов и CPU-hotplug механизма и позволяющее динамически включать и отключать ядра процессора. Система, предложенная авторами, представляет из себя 3 больших модуля:

1. *Online Resource Usage Monitor* — модуль ответственный за сбор информации о состоянии каждого ядра процессора. Собирает такие данные как частота, загруженность, температура (если ядро поддерживает).

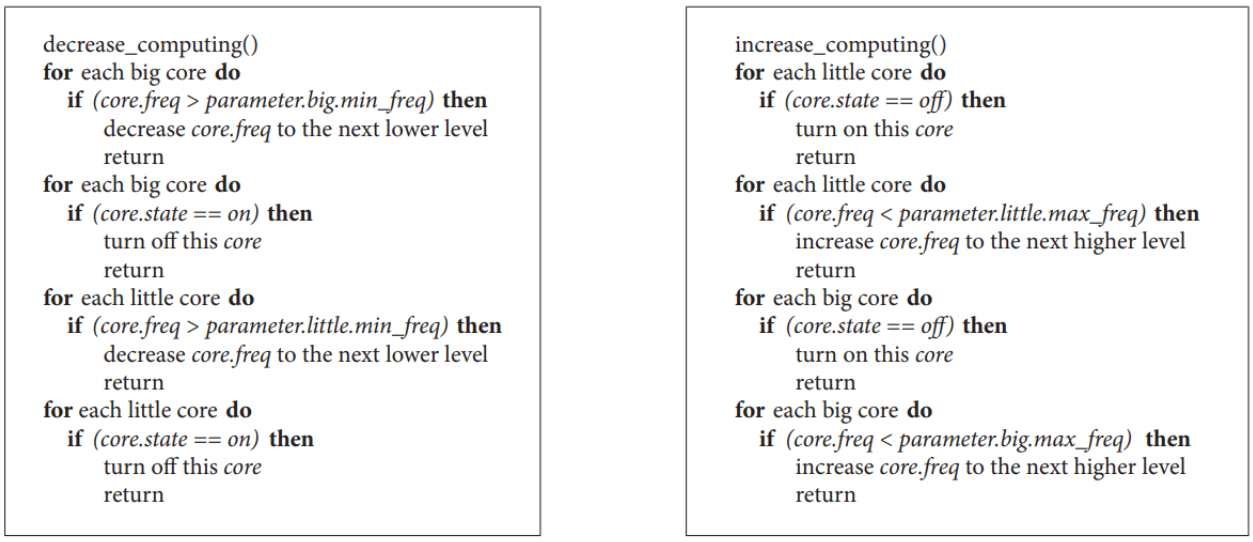
2. *Policy Manager* — модуль, позволяющий более тонко настраивать данный механизм. Например, выставлять ограничения на диапазон частот, которые могут использоваться, выбирать базовый регулятор от работы которого будет отталкиваться система или существенно влиять на алгоритм, который использует контроллер.

3. *Low-Power Controller* — занимается изменением частоты, напряжения и включением / отключением ядер. Алгоритм его работы заключаются в вызове одной из двух рабочих процедур, представленных на рисунке 1. Решение о вызове той или иной процедуры принимается на базе предсказаний рабочей нагрузки при помощи скользящего среднего.

Данный подход позволяет уменьшить потребление энергии смартфоном от 22% до 79% в зависимости от типа нагрузки. Наилучшие результаты были получены при записи видео на телефон.

2.2.3. Машинное обучение

В статье [22] используется более комплексный подход к оптимизации энергопотребления, основанный на идее о том, что оптимизировать



```
decrease_computing()
for each big core do
  if (core.freq > parameter.big.min_freq) then
      decrease core.freq to the next lower level
      return
for each big core do
  if (core.state == on) then
      turn off this core
      return
for each little core do
  if (core.freq > parameter.little.min_freq) then
      decrease core.freq to the next lower level
      return
for each little core do
  if (core.state == on) then
      turn off this core
      return
```

```
increase_computing()
for each little core do
  if (core.state == off) then
      turn on this core
      return
for each little core do
  if (core.freq < parameter.little.max_freq) then
      increase core.freq to the next higher level
      return
for each big core do
  if (core.state == off) then
      turn on this core
      return
for each big core do
  if (core.freq < parameter.big.max_freq) then
      increase core.freq to the next higher level
      return
```

Рис. 1: Рабочие процедуры алгоритма [15]

надо не только какой-то один уровень системы (например уровень ОС), а все, начиная от приложения, заканчивая аппаратным уровнем. Однако, в тоже время, в данной работе описан интересный DVFS регулятор который лежит в центре всех проводимых оптимизаций.

Основная идея в том, что программист, при написании программы, собственноручно разбивает её на блоки (фреймы) при помощи специальных аннотаций в коде, в которых указывается тип нагрузки (может не указываться, но с известным типом оптимизатор работает значительно лучше) и желаемая производительность, измеряемая в количестве кадров. Эта информация передается регулятору, который состоит из двух частей:

1. Модуля предсказания рабочей нагрузки, который по данным о текущих фреймах считает ожидаемую нагрузку на систему при их исполнении и $t_{deadline}$— время их желательного завершения, используя при этом алгоритм AEWMA (Exponential Weighted Moving Average) с адаптивными весами. Причем нагрузка предсказывается отдельно для каждой группы фреймов объединенных по типу нагрузки.

2. Модуля определения оптимального напряжения и частоты, кото-

рый принимает предсказания нагрузки из предыдущего модуля и по ним определяет оптимальное напряжение и частоту при помощи Q-Learning алгоритма (Обучения с подкреплением).

После установки P-состояния системы происходит оценка эффективности принятого решения, а именно оценивается выполнился ли фрейм вовремя или его $t_{deadline}$ было превышено. В зависимости от результатов корректируется работа алгоритма Q-Learning для следующих предсказаний.

Таким образом, система моделирует поведение операционной системы мягкого времени, пытаясь выполнить набор задач, просрочив при этом как можно меньшее количество дедлайнов на время их выполнения.

Такой подход позволяет экономить до 30% энергии при просмотре видео, а также до 60% энергии при работе с приложениями, поддерживающими программное разбиение на фреймы. При этом потери производительности составляют порядка 1.5%

В другой работе [3], также использующей машинное обучение, применяются совместно две технологии: DVFS регулятор и Dynamic Core Selection (DCS). Также, в отличие от большинства других решений, представленный в статье подход оптимизирует не только расход энергии, идущей на полезную работу системы, но и учитывает потери энергии идущие на нагрев процессора, которые определяются текущей температурой устройства и его платформозависимыми константами.

Для оптимизации представлен алгоритм базирующийся на Q-Learning. На вход он принимает температуру и суммарное количество циклов ядер процессора (называемое состоянием) с момента последней итерации алгоритма. Далее предложенный подход действует следующим образом:

1. Оценивается то, насколько предсказанная на предыдущей итерации алгоритма производительность отличается от реальной.

2. На основе этой информации обновляются значения в Q-table — таблице, которая сопоставляет состоянию и текущей конфигура-

ции (тоесть текущей частоте и набору включенных ядрер) системы новую конфигурацию, в которую должна перейти система.

3. Предсказывается следующее вероятное состояние системы при помощи EWMA.

4. По предсказанному состоянию и таблице выбирается оптимальная конфигурация для перехода.

Представленный подход позволяет в среднем улучшить энергоэффективность на 22% в сравнении с базовыми регуляторами, при этом теряя в производительности около 5% для большинства тестируемых приложений.

В [31] описан один из наиболее популярных подходов к оптимизации энергопотребления при помощи машинного обучения, а именно описано использование алгоритма AEWMA для предсказания рабочей нагрузки и Q-Learning для выбора оптимального состояния системы (частоты) по предсказанной нагрузке.

Однако важной отличительной особенностью указанного подхода является адаптация алгоритма к резким сменам типа рабочей нагрузки. Для этого был модифицирован классический AEWMA алгоритм в модификацию AEWMA-MSE. Работает она следующим образом:

1. В момент вызова рабочей процедуры оценивается качество предсказания на предыдущей итерации при помощи метрики MSE.

2. На основании величины этой метрики обновляются коэффициенты в скользящем среднем.

3. Если MSE показатель достаточно большой (произошло резкое изменение рабочей нагрузки), сбрасывается текущее состояние Q-Learning алгоритма (обнуляется Q-table) и заново происходит его переобучение.

Данный подход позволяет минимизировать энергопотребление смартфона до 29% в сравнении такими регуляторами как ondemand и interactive.

2.2.4. Предварительные вычисления

В работе [32] показана корреляция между энергопотреблением и специальной аппаратной характеристикой — $Operational\,Intensity$, а также предложен DVFS регулятор, построенный на базе данной характеристики.

Основная идея заключается в следующем:

1. Введена метрика OI — $Operational\,Intensity$, вычисляемая следующим образом:

$$OI = \frac{The\ number\ of\ Operations}{BUS\ access \times Byte}.$$

2. Построена таблица соответствия между IO и оптимальной частотой процессора.

3. Итоговый алгоритм заключается в том, что во время исполнения рабочей процедуры регулятора вычисляется IO, а затем находится оптимальная частота по построенной таблице.

Такой подход позволяет экономить на 8-9% больше энергии чем такие регуляторы, как performance и ondemand.

В статье [16] представлено решение, которое, в отличие от большинства других подходов, не решает проблему энергопотребления в общем, а позволяет оптимизировать лишь конкретное приложение под конкретную платформу, существенно уменьшив энергопотребление без потери производительности. Также интересен способ реализации представленный в работе: здесь управление частотой и напряжением происходит через базовый регулятор UserSpace, а не реализован собственный scaling governor, как это происходит в большинстве других решений. Процесс оптимизации можно разделить на 2 стадии, описанные ниже.

Первая стадия — offline profiling, она заключается в следующем: оптимизируемое приложение запускается на нескольких комбинациях пар (частота процессора, скорость памяти) и замеряются различные ключевые метрики работы приложения при каждой такой конфигурации, как

правило это потребленная мощность и увеличение производительности относительно самой слабой комбинации. Затем полученные значения интерполируются по всем возможным комбинациям частоты и скорости памяти. Также на этом этапе производится запуск приложения с регулятором по умолчанию и оценивается его средняя производительность — R, которой система будет добиваться во время работы данного приложения.

Вторая стадия — online controlling, на которой и происходит выбор оптимальной частоты. Во время этой стадии в каждый момент времени оценивается разница между текущей производительностью и желаемой производительностью R, посчитанной во время профилирования. Затем оценивается как нам из текущего состояния частоты процессора и памяти перейти в такое, чтобы достичь R, а именно выбирается такой набор смены состояний, чтобы переключение между ними было оптимальным с точки зрения производительности и затраченного времени.

Авторы провели такую оптимизацию, выбрав 6 популярных, по их мнению, приложений, которые оказывают разного типа нагрузки на систему. Результаты оптимизации можно видеть в таблице 1, где Performance и Energy показывают выигрыш в процентах у базового регулятора по производительности и энергопотреблению соответственно.

Application Name	Performance	Energy
VidCon	-0.4%	25.3%
MobileBench	4.1%	15.3%
AngryBirds	0.6%	14.9%
WeChat Video Call	-0.4%	27.2%
MX Player	0.0%	4.2%
Spotify	9.3%	31.6%

Таблица 1: Результаты работы алгоритма [16]

2.2.5. Выводы

Таким образом все найденные в рамках данной работы статьи удалось классифицировать на 3 большие группы: адаптивные алгоритмы, алгоритмы, использующие машинное обучение и алгоритмы, для работы которых необходимо заранее проводить какие-либо предварительные вычисления.

Важно отметить, что ни одна из групп не является наиболее выигрышной в плане экономии энергии в сравнении с остальными. Наилучшие результаты в каждом классе алгоритмов попадают в диапазон — 30-40% сэкономленной энергии в сравнении с регуляторами из ОС Android при тестировании на наиболее обобщенных типах рабочих нагрузок.

Также проведенный в рамках данной работы обзор показал, что методы стохастической оптимизации не применялись для регулировки частот смартфона, хотя во многих схожих сферах они активно используются, например, в работе [39] описывается использование подобного математического аппарата для регулировки напряжения в MP3 плеере.

3. Инфраструктура для загрузки алгоритмов DVFS

Существенной частью работы стал выбор смартфона, на котором будет апрбирован итоговый алгоритм DVFS, а также создание инфраструктуры для загрузки таких алгоритмов на выбранное мобильное устройство. Данная глава содержит обзор выбранного устройства — Xiaomi Redmi Note 8 pro и более подробное описание организованной инфраструктуры.

3.1. Xiaomi Redmi Note 8 Pro

В качестве тестового стенда для апробации алгоритма был выбран смартфон под названием Xiaomi Redmi Note 8 Pro [36] со следующими ключевыми характеристиками:

- 6 гигабайт оперативной памяти,
- 64 гигабайта основной памяти,
- ОС Android 9 с оболочкой MIUI-11, которая впоследствии была обновлена до Android 10,
- 8-ми ядерный процессор Mediatek Helio G90T.

Выбор данного устройства в первую очередь обусловлен высокой популярностью этого смартфона в России [25], а значит наличием большого количества материалов по работе с ним. Вдобавок смартфон позволяет устанавливать последние версии операционной системы Android, что делает данную выпускную квалификационную работу более актуальной, а также имеет приемлемую цену и современный многоядерный процессор [24].

3.2. Права суперпользователя

Одной из первоочередных задач стало получение прав суперпользователя на выбранном смартфоне, так как они необходимы для уста-

новки новых прошивок и переключения между различными DVFS-регуляторами. Для этого были изучены материалы доступные в сети [1, 37] и на их основе проделаны следующие шаги:

1. Разблокирован загрузчик смартфона при помощи официальной утилиты Xiaomi — Mi unlock tool [23].

2. Установлено стороннее рекавери TWRP [34] путем перевода телефона в bootmode и использования утилиты platform-tools [10].

3. При помощи TWRP отключен Android Verified Boot 2.0, после чего установлен magisk manager и тем самым получены права суперпользователя.

3.3. Особенности работы со сторонними ядрами и прошивками

Важной частью создания инфраструктуры стал поиск, сборка и установка на смартфон стороннего ядра с открытым исходным кодом, что, в будущем, позволило добавлять и запускать свои собственные алгоритмы регулировки частот на используемом смартфоне.

В рамках работы были изучены официальные ядра из репозитория Xiaomi [26], однако, после нескольких неудачных попыток их собрать, был найден другой проект с открытым исходным кодом — AgentFabulous begonia kernel project [5], исходники которого удалось адаптировать для сборки компилятором Proton Clang [19]. Данный компилятор специально создан для разработчиков ядер и, по заверению авторов [19], значительно проще и удобнее в использовании, чем подобные инструменты от компании Google [4]. Результатом работы на данном шаге стала инструкция по сборке и настройки найденного ядра для используемого в работе смартфона

Однако, найденное ядро оказалось несовместимым с официальной прошивкой Xiaomi — MIUI 12, так как начиная с версии Android 10 упомянутая компания начала бороться с возможностью восстановления телефона вне сервисного центра в случае его полной поломки(hardbrick),

в связи с чем прошивки поделились на три класса: CFT, NON-CFT и GSI. CFT прошивки поддерживают возможность восстановления системы в обход новых ограничений, однако требуют поддержки этой технологии от ядра, что делает несовместимыми такие ядра с остальными прошивками.

Ядро из проекта AgentFabulous begonia kernel project разработано для CFT прошивок, поэтому было принято решение установить такую прошивку на смартфон. Для этого была выбрана прошивка POSP — Potato open source project [29]. Она наиболее близка к реализации Android в чистом виде — проекту AOSP [6], однако разработана непосредственно для используемого нами устройства — Xiaomi Redmi Note 8 pro. К тому же данный проект ведется теми же авторами, что и используемое ядро, таким образом найденная прошивка должна обеспечивать наилучшую их совместимость.

Для установки POSP были проделаны следующие шаги:

1. С сайта проекта был скачан образ прошивки под наше устройство, распакован и загружен на microSD карту.

2. При помощи TWRP рекавери были отформатированы разделы: Dalvik cache, Data и Cache, а также проведен вайп памяти всего телефона.

3. Затем с microSD карты установлен сам образ прошивки и отключен Android Verified Boot 2.0.

4. На новой прошивке заново были получены права суперпользователя при помощи установки системного приложения magisk [2].

4. Предлагаемый алгоритм DVFS

Многие процессы в нашем мире имеют случайную, стохастическую, и очень сложную природу, а средства наблюдения за ними не идеальны и могут вносить свои погрешности в наблюдение и изучение таких процессов. В этой связи в современном мире все большее и большее распространение приобретают алгоритмы стохастической аппроксимации, которые, несмотря на перечисленные сложности, позволяют эффективно наблюдать и оценивать такие процессы.

В данной главе вводится необходимый математический аппарат для применения одного из таких алгоритмов — SPSA в сфере регулировки частот смартфона, а также описывается итоговое поведение разработанного DVFS-регулятора.

4.1. Стохастическая аппроксимация со случайными направлениями

Пусть существует некоторый эмпирический функционал качества — $F(x, w)$, который мы можем подсчитывать на определенных временных интервалах, x — изучаемый параметр, w — случайный вектор. Тогда сам функционал качества возможно оценить как $f(x) = E\{F(x, w)\}$ и тем самым можно решать задачу оптимизации функционала среднего риска:

$$f(x) = E\{F(x, w)\} \to \min_x .$$

Однако с течением времени оптимальное значение обычно дрейфует, поэтому наряду с оптимизацией важной проблемой становится задача трекинга — отслеживание изменяющейся оптимальной точки экстремума, более формально задача ставится следующим образом.

Пусть $F_t(x, w)$ — случайная функция от дискретного времени t, параметра x и случайного вектора w. Обозначим "текущий" функционал среднего риска, как:

$$f_t(x) = E_w\{F_t(x, w)\}$$

и точку минимума $f_t(x)$:

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

Тогда задача состоит в том, чтобы по наблюдению за случайными величинами $F_t(x_n, w_n)$, n = 1, 2,.. построить последовательность оценок $\{\hat{\theta}_n\}$ такую что $||\hat{\theta}_n - \theta_t|| \to \min$

Пробным одновременным возмущением будем называть последовательность наблюдаемых одинаково симметрично распределенных случайных векторов $\{\Delta_n\}$ с матрицами ковариаций:

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

где $\delta_{nj} \in \{0, 1\}$ — символ Кронекера. $0 < \sigma_\Delta < \infty$. На практике обычно в качестве пробного одновременного возмущения используют бернуллиевские случайные вектора (координаты вектора Δ_n не зависят друг от други и принимают значения ± 1 с равной вероятностью).

Доказано [38], что при зашумленных наблюдениях без существенных потерь в скорости сходимости для построения состоятельной последовательности оценок точки минимума функционала $f(x)$ можно воспользоваться алгоритмом:

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

где используются два зашумленных наблюдения:

$$y_n^+ = F(\hat{\theta}_{n-1} + \beta_n \Delta_n, w_n^+) + v_n^+, y_n^0 = F(\hat{\theta}_{n-1}, w_n^0) + v_n^0.$$

Этот алгоритм и называется стохастической аппроксимацией со случайными направлениями или в англоязычной литературе — Simultaneous perturbation stochastic approximation (SPSA). Также схожим поведением обладает и его вариация с одним зашумленным измерением:

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

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

В обоих алгоритмах $\{\alpha_n\}$ и $\{\beta_n\}$ — последовательности неотрицательных чисел, удовлетворяющие следующим условиям:

$$\sum_n \alpha_n = \infty, \beta_n \to \infty, \sum_n \frac{\alpha_n^2}{\beta_n^2} < \infty, \sum_n \alpha_n \beta_n < \infty.$$

Однако в некоторых случаях вместо данных последовательностей могут выбираться константные значения α и β, тогда представленный алгоритм несколько лучше решает задачу трекинга, но сходится к оптимальной оценке хуже [14].

4.2. Модель состояния системы

В ходе работы была разработана модель, которая представляет из себя функционал, описывающий состояние системы в зависимости от поступающих на смартфон нагрузок и используемой им частоты в конкретный момент времени.

Определим метрику рабочей нагрузки за период времени τ, при используемой частоте f следующим образом:

$$workload_\tau(f) = 1 - \frac{idle\_time_\tau(f)}{\tau},$$

где $idle\_time_\tau(f)$ — время простоя процессора на временном промежутке τ с частотой f.

Обозначим $table(f)$ — порядковый номер частоты f в таблице частот процессора.

Тогда итоговая модель определяется следующим образом:

$$F_\tau(f) = 2^{((\text{workload}_\tau(f) - \lambda)/2)} + \gamma 1.5^{\text{table}(f)},$$

где λ — эмпирически выверенный порог рабочей нагрузки (порядка 60 %), γ — параметр, отвечающий за чувствительность системы к той или иной используемой частоте.

Используемую модель логически можно поделить на 2 части: первое слагаемое является штрафом за избыточную нагрузку при используемой частоте, а второе определяет стоимость использования процессо-

ром конкретной частоты.

Таким образом задача определения оптимальной частоты системы сводится к поиску дрейфующей точки экстремума функционала среднего риска $g(f) = E\{F_\tau(f)\}$.

4.3. Алгоритм DVFS

Для поиска минимума функционала $g(f)$ воспользуемся стохастической аппроксимацией со случайными направлениями, описанной в главе 4.1, в частности его вариацией с одним измерением. Обозначим множество частот доступных процессору — $Freq$. В качестве последовательностей $\{\alpha_n\}$, $\{\beta_n\}$ примем константные значения α и β.

Таким образом итоговый алгоритм работы системы выглядит следующим образом:

1. Выбираем начальное значение оценки: $\hat{f}_0$.
2. Генерируем очередной элемент случайной последовательности чисел, равных ± 1 с одинаковой вероятностью — Δ_n.
3. Возмущаем текущую оптимальную оценку: $f_n = \mathscr{P}(\hat{f}_{n-1} + \beta\Delta_n)$, где $\mathscr{P}(\cdot)$ — проектор в множество $Freq$.
4. Запускаем систему на f_n.
5. Получаем зашумленное наблюдение: $y_n = F_\tau(\hat{f}_{n-1} + \beta\Delta_n) + v_n$.
6. Обновляем текущую оптимальную оценку: $\hat{f}_n = \mathscr{L}(\hat{f}_{n-1} - \frac{\alpha}{\beta}\Delta_n y_n)$, где $\mathscr{L}(\cdot)$ — проектор в отрезок $[\min(Freq), \max(Freq)]$.
7. Переходим к шагу 2.

5. Особенности реализации алгоритма

Представленный алгоритм DVFS был реализован в качестве регулятора масштабирования подсистемы CPUFreq ОС Android. Реализация была встроена в ядро Android 10 (v 4.14.193) и апробирована на смартфоне Xiaomi Redmi Note 8 pro. В следующих подглавах описаны ключевые особенности указанной реализации.

5.1. Поддержка многоядерности

Важной особенностью современных мобильных процессоров является их гетерогенность, как правило она обусловлена наличием двух вычислительных кластеров: энергоэффективного, содержащего менее производительные — "малые" ядра, и производительного, ядра которого обеспечивают высокую вычислительную мощь, однако потребляют значительно больше энергии.

Процессор Mediatek Helio G90T [24] не является исключением, он содержит два высокопроизводительных ядра архитектуры Arm Cortex-A76 и шесть менее мощных, но более энергоэффективных ядер — Arm Cortex-A55.

В связи с этим представляемая реализация в ядре операционной системы регистрирует две независимые копии алгоритма: по одной на каждый вычислительный кластер. Такой подход позволяет отдельно настраивать поведение конкретного кластера, а также, путем настройки параметров модели, приоритезировать использование более энергоэффективных кластеров.

В тоже время последние версии мобильных процессоров обладают передовой трехкластерной архитектурой [33] по типу «1 + 3 + 4». Хочется отметить что предлагаемый в работе подход легко масштабируется и на такие, более современные, архитектуры.

5.2. Поддерживаемые настройки

В ходе работы также часть параметров модели и настроек алгоритма были вынесены непосредственно в операционную систему Android, что позволяет корректировать поведение алгоритма прямо во время его работы, при наличии прав суперпользователя. Далее идет более подробное описание выделенных настроек.

- beta: параметр SPSA, регулирующий величину возмущения оптимальной оценки на каждой итерации алгоритма;
- alpha: второй параметр SPSA, позволяющий регулировать скорость сходимости алгоритма к оптимальной оценке;
- gamma: параметр модели, отвечающий за "стоимость" использования конкретной частоты процессором;
- up_threshold: порог рабочей нагрузки, при котором система начинает получать штрафы за избыточную нагрузку;
- sampling_rate: частота обновления оценки, предоставляемой алгоритмом;
- sampling_down_factor: множитель, позволяющий регулировать частоту обновления оценки при преодолении up_threshold (по умолчанию 1).

Все описанные настройки вынесены в каталог: /sys/devices/system/cpu/cpufreq/spsa

6. Тестирование

Ключевой частью работы стала проверка, насколько хорошо представленный алгоритм справляется со своей задачей: выбором наиболее энергоэффективной частоты, при этом не снижая производительность смартфона. Для этого автором работы совместно с Александром Божнюком были выбраны инструменты для проведения тестов и определен набор тестовых случаев для проверки эффективности алгоритмов DVFS. Позже, непосредственно самим автором работы, было проведено тестирование разработанного алгоритма и анализ полученных результатов.

6.1. Инструменты и критерии тестирования

Monkeyrunner

Запуск тестов проводился при помощи официальной утилиты от компании Google — Monkeyrunner [9], которая предоставляет API для написания программ на языке Python. Эти программы позволяют управлять Android устройством вне конкретного приложения, а удаленно, при помощи компьютера, используя технологию Android debug bridge (ADB).

Такой подход позволяет эмулировать взаимодействие пользователя с устройством: имитировать касания экрана, запускать или удалять приложения. При этом, в отличие от многих утилит [8, 13], Monkeyrunner не требует доступа к исходному коду тестируемого приложения. Данная утилита поставляется вместе с SDK.

Тестовые случаи

В ходе работы для проведения тестирования алгоритмов DVFS был выделен ряд тестовых случаев — возможных сценариев использования смартфона. В первую очередь были изучены сценарии тестирования в работах из главы 2.2, на основе которых удалось выделить пять основных тестовых случаев. Часть из них имитирует сложные вычислитель-

ные задачи, дающие высокую нагрузку на процессор, например, игры или съемка видео, а часть, наоборот, позволяет оценить поведение алгоритма DVFS на более "простых" задачах для процессора, таких как просмотр видео или набор текстовых сообщений.

Ниже приведено более подробное описание, пяти итоговых тестов:

1. **Camera test:** данный тест запускает камеру смартфона, после чего в течение 15 минут ведет съемку видео.

2. **Typing test:** устанавливает на смартфон приложение Notes, если оно не было установлено до этого, затем создает в нем новую заметку и в течение 15 минутного периода производит набор теста в открытую заметку.

3. **Flappy bird test:** при необходимости производит установку игры Flappy Bird на устройство, запускает ее и на протяжении 15 минут имитирует игровую деятельность при помощи касаний экрана смартфона.

4. **Xtream test:** тест аналогичен предыдущему, однако запускается более требовательная к производительности смартфона игра — Trial Xtreme 3.

5. **Video test:** устанавливает приложение VLC player for Android и при помощи него воспроизводит 15-ти минутное видео.

Указанные тесты были реализованы в качестве скриптов на языке Python Александром Божнюком, как часть его учебной практики. Автором использовались уже готовые версии программ [12].

6.2. Методология тестирования

Подготовка смартфона

Для обеспечения наиболее объективного результата выполнения тестов в ходе замеров со смартфоном проводились следующие действия:

- были удалены все сторонние приложения, не относящиеся к проводимым тестам;
- все тесты проводились в авиарежиме, таким образом были отключены Wi-Fi, GPS, Bluetooth и другие периферийные устройства;
- между тестами выдерживалась двух минутная пауза, чтобы перед новым тестом все фоновые процессы связанные с предыдущим корректно завершились, а само устройство остыло до исходной температуры.

Оценка потребляемой энергии:

Подсистема CPUFreq предоставляет информацию о времени проведенном кластером процессора на конкретной частоте за все время использования устройства, а также механизмы сброса и возобновления подсчета данной статистики. Во время тестирования, перед каждым тестом, статистика использования частот сбрасывалась, а после завершения теста извлекалась и сохранялась на компьютер. Каждый из описанных тестов был проведен по 4 раза, итоговые средние значения представлены в таблице 5.

Для подсчета потребляемой смартфоном энергии был извлечен его *powerprofile.xml* файл, который предоставляется производителем устройства и содержит информацию о энергопотреблении смартфона и его периферии. В частности, данный файл содержит константы энергопотребления кластеров процессора, при их работе на той или иной частоте в течение 10 миллисекундного интервала. Эти значения и были использованы в работе для подсчета энергии, они представлены в таблице 2.

Таким образом для итоговой оценки энергопотребления в миллиампер часах была использована следующая формула:

$$power\_consumption = \sum_{Freq} \frac{time\_on\_freq(i) * consumption(i)}{3600 * 100},$$

где $time\_on\_freq(i)$ — время проведенное на частоте i из таблицы 5,

A55		A76	
Частота (в Гц)	Энергопотребление (в мА)	Частота (в Гц)	Энергопотребление (в мА)
2000000	90.04	2050000	324.33
1933000	85.8	1986000	307.98
1866000	80.27	1923000	291.52
1800000	72.77	1860000	269.61
1733000	66.61	1796000	247.53
1666000	62.05	1733000	233.56
1618000	58.95	1670000	209.73
1500000	52.33	1530000	177.39
1375000	44.83	1419000	152.46
1275000	39.69	1308000	130.33
1175000	35.5	1169000	105.19
1075000	31.24	1085000	91.11
975000	27.86	1002000	79.53
875000	25	919000	70.65
774000	23.5	835000	61.38
500000	19.55	774000	56.85

Таблица 2: Xiaomi Redmi Note 8 Pro константы энергопотребления

$consumption(i)$ — энергопотребление i-й частоты за 10-ти миллисекундный интервал из таблицы 2.

Оценка производительности:

Важно не только проверить, что представленный алгоритм энергоэффективен, но и оценить производительность смартфона при его использовании, так как действительно эффективный алгоритм не должен влиять на взаимодействие пользователя с устройством.

Для комплексной оценки производительности было принято решение использовать один из наиболее популярных тестов — AnTuTu benchmark [27]. Данная утилита после ряда замеров выставляет баллы устройству по 5 различным категориям: CPU, GPU, Memory, Ux и Total. Таким образом можно наблюдать как использование различных CPUFreq регуляторов влияет на поведение смартфона. Также ключевым фактором выбора данной утилиты стало то, что она поддерживает тестирование устройств с процессорами Mediatek, в отличие от многих других популярных аналогов [28]. Ниже представлены результаты работы утилиты для различных алгоритмов DVFS:

Регулятор	Всего	CPU	GPU	Memory	UX
Powersave	128356	28883	50193	30792	18458
SPSA	244251	80555	74357	40897	48442
OnDemand	251952	83704	76335	42020	49893
interactive	245154	79459	76880	39403	49412
Performance	260891	86756	79952	42364	51789

Таблица 3: Баллы AnTuTu benchmark

6.3. Анализ результатов

Финальные результаты тестирования энергопотребления представлены в таблице 4. В ней, в первой колонке, указаны результаты представленного в данной работе алгоритма — SPSA, а в двух других результаты наиболее часто используемых в современных смартфонах регуляторов — OnDemand и Interactive. В процентах указано насколько

SPSA более (значение с плюсом) или менее (значение с минусом) энергоэффективен.

Тест	SPSA (в мАч)	OnDemand (в мАч)	Interactive (в мАч)
Camera test	57.119	70.918 (+19.46%)	44.823 (-21.6%)
Typing test	77.187	75.536 (-2.14%)	84.306 (+8.4%)
Trial Xtreme 3	34.100	40.066 (+14.89%)	35.801 (+4.7%)
Flappy bird	27.690	24.643 (-11.00%)	43.710 (+36.6%)
Video test	31.723	25.736 (-18.87%)	47.660 (+33.4%)

Таблица 4: Итоговые результаты тестирования

Из представленной таблицы следует, что SPSA справляется с регулировкой частот не хуже, чем используемые на данный момент подходы в ядре ОС Android, а на некоторых тестах (Trial Xtreme 3) выигрывает оба конкурирующих решения. Также замеры производительности смартфона, указанные в таблице 3, показывают, что решение эффективно и не оказывает существенного влияния на взаимодействие пользователя с устройством.

Таким образом алгоритмы стохастической оптимизации, в частности стохастическая аппроксимация со случайными направлениями, могут быть успешно применены для регулировки частот смартфона, не уступая существующим на рынке решениям.

Заключение

В рамках данной дипломной работы были достигнуты следующие результаты.

1. Проведен обзор существующих подходов к оптимизации энергопотребления, для этого:
   - изучены регуляторы: Performance, PowerSave, UserSpace, Ondemand, Conservative и Interactive;
   - рассмотрены современные подходы: адаптивные алгоритмы, алгоритмы машинного обучения и алгоритмы, использующие предварительные вычисления.
2. В качестве устройства для апробации выбран смартфон Xiaomi redmi note 8 pro, для него создана инфраструктура загрузки алгоритмов DVFS: собрано и прошито ядро, выбрана и установлена прошивка, поддерживающая установку сторонних ядер Android.
3. Разработан алгоритм DVFS с использованием стохастической оптимизации:
   - создана модель, описывающая зависимость состояния смартфона от поступающей нагрузки и используемой им частоты;
   - на базе модели реализован алгоритм стохастической аппроксимации и встроен в ядро ОС Android [11].
4. Проведено тестирование полученного DVFS-алгоритма по 5 тестовым случаям, как итог: предлагаемый алгоритм не уступает по эффективности таким популярным регуляторам как OnDemand и Interactive.

Список литературы

[1] 4pda forum. Custom roms for Xiaomi redmi note8 pro discussion. URL: https://4pda.ru/forum/index.php?showtopic=965629.

[2] 4pda forum. Magdisk open source utility discussion. URL: https://4pda.ru/forum/index.php?showtopic=774072.

[3] A. Das, M. J. Walker, A. Hansson, B. M. Al-Hashimi, and G. V. Merrett, "Hardware-software interaction for run-time power optimization: A case study of embedded Linux on multicore smartphones," in Proceedings of the International Symposium on Low Power Electronics and Design, Jul 2015.

[4] AOSP Clang. Official Android kernel compiler prebuilts for kernel developers. URL: https://android.googlesource.com/platform/prebuilts/clang/host/linux-x86/.

[5] Agent Fabulous kernel sources. URL: https://github.com/AgentFabulous/begonia.

[6] Android source. Android Open Source Project (AOSP) starting page. URL: https://source.android.com.

[7] D. Snowdon, S. Ruocco, and G. Heiser, "Power management and dynamic voltage scaling: Myths and facts," vol. 12, pp. 1–7, Jan 2005.

[8] Developers Android. Espresso testing utility for Android devices. URL: https://developer.android.com/training/testing/espresso.

[9] Developers Android. Monkeyrunner tool overview page. URL: https://developer.android.com/studio/test/monkeyrunner.

[10] Developers Android. SDK Platform Tools release notes. URL: https://developer.android.com/studio/releases/platform-tools.

[11] Github. Repository Spsa Android module sources. URL: https://github.com/jackbogdanov/DVFS-for-begonia/tree/android-10.0.

[12] Github. Repository with test sources for DVFS algorithms testing. URL: https://github.com/bozhnyukAlex/dvfs_testing.

[13] Github. Robotium — Android test automation framework sources. URL: https://github.com/RobotiumTech/robotium.

[14] Granichin O. N. Linear regression and filtering under nonstandard assumptions (Arbitrary noise) // Trans. on Automatic Control. 2004. Vol. 49. Oct. №10. P. 1830-1835.

[15] J. Choi, B. Jung, Y. Choi, and S. Son, "An Adaptive and Integrated Low-Power Framework for Multicore Mobile Computing," Mobile Information Systems, Jun 2017.

[16] K. Rao, S. Yalamanchili, Y. Wardi, J. Wang, H. Ye, "Application-Specific Performance-Aware Energy Optimization on Android Mobile Devices," in 2017 IEEE International Symposium on High Performance Computer Architecture, pp. 1-12, 2017.

[17] Kushner H.J., Yin G.G. Stochastic Approximation Algorithms and Applications. New York: Springer–Verlag. 1997. 415 p.

[18] L. Broyde, K. Nixon, X. Chen, H. Li, and Y. Chen, "MobiCore: An adaptive hybrid approach for power-efficient CPU management on Android devices," in 2017 30th IEEE International System-on-Chip Conference (SOCC), pp. 221–226, Sep 2017.

[19] LLVM and Clang compiler toolchain built for kernel development. Github profile. URL: https://github.com/kdrag0n/proton-clang.

[20] The Linux Kernel documentation. CPU Performance Scaling page. URL: https://www.kernel.org/doc/html/v4.15/admin-guide/pm/cpufreq.html.

[21] Linux kernel official documentation. CPU frequency and voltage scaling code in the Linux(TM) kernel. URL: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[22] Luis Alfonso Maeda-Nunez, "System-level power management using online machine learning for prediction and adaptation, University of Southampton," in Faculty of Physical Sciences and Engineering, Doctoral Thesis, pp. 1-155, Jul 2016.

[23] MIUI official cite. Unlocking bootloader instruction. URL: http://www.miui.com/unlock/index_en.html.

[24] Mediatek official cite. MediaTek Helio G90 Series page. URL: https://www.mediatek.com/products/smartphones/mediatek-helio-g90-series.

[25] Mobile Vendor Market Share Russian Federation. Official cite. URL: https://gs.statcounter.com/vendor-market-share/mobile.

[26] Official Xiaomi kernel sources. URL: https://github.com/MiCode/Xiaomi_Kernel_OpenSource.

[27] Official cite of benchmark for mobile decices — Antutu benchmark. Main page. URL: https://www.antutu.com/en/index.htm.

[28] Official cite of benchmark for mobile decices — Geekbench 5. URL: https://www.geekbench.com/.

[29] Potato open source project. Official cite. URL: https://potatoproject.co/.

[30] Previous researchers results. Google tables. URL: https://docs.google.com/spreadsheets/d/1JDhrmRp6oMgL_sZcg5oePtcB_Q

[31] S. A. Carvalho, D. C. Cunha, and A. G. Silva-Filho, "Autonomous Power Management for Embedded Systems Using a Non-linear Power Predictor," in 2017 Euromicro Conference on Digital System Design (DSD). IEEE, Sep 2017.

[32] S. J. Cho, S. H. Yun, and J. W. Jeon, “A powersaving DVFS algorithm based on operational intensity for embedded systems,” pp. 1–6, Feb 2015.

[33] Snapdragon 888 5G Mobile Platform. Official processor page. URL: https://www.qualcomm.com/products/snapdragon-888-5g-mobile-platform.

[34] TeamWin - TWRP recovery. Official cite. URL: https://twrp.me.

[35] XDA forum for Android custom roms developers. Android governers overview. URL: https://forum.xda-developers.com/nexus-4/general/guide-android-governors-explained-t2017715.

[36] Xiaomi official cite. Xiaomi redmi note8 pro page overview. URL: https://www.mi.com/global/redmi-note-8-pro.

[37] YouTube k k techno video chanel. How To MIUI 12 | Root With Magisk Manager install | Android 10 | Redmi note 8 pro |. URL: https://youtu.be/kIqDP8t3wZA.

[38] Граничин О. Н., Поляк Б. Т. Рандомизированные алгоритмы оценивания и оптимизации при почти произвольных помехах. М.: Наука. 2003. 291 с.

[39] Краснощеков В. Е. Рандомизированный алгоритм для оптимизации энергопотребления в мобильных устройствах. : Стохастическая оптимизация в информатике. 2006. 204 с.

	Camera test		Typing test		Trial Xtreme 3		Flappy bird		Video test	
Freq (Гц)	OnDemand	SPSA	OnDemand	SPSA	OnDemand	SPSA	OnDemand	SPSA	OnDemand	SPSA
2000000	3528.25	17169	1586.5	16199.25	296.75	7458.5	155.25	1459.25	200	5648.75
1933000	3513.75	9116.5	1325.25	8434.75	182	3994.75	25.5	778	53.75	3158.75
1866000	4789.75	3900.25	2301.5	2937.25	470.75	1560	45.75	298.5	83.75	1100.25
1800000	6363.25	3269.5	3116.75	2525.5	1117	1467.75	59.5	250.5	91.75	1030.25
1733000	7782.25	3251.25	3976.5	2666	2312.25	1752.5	78.75	328.75	154.75	1197
1666000	6388.5	2446	2855	1968.75	2166.25	1410	89.5	227.25	139.25	911.5
1618000	18415.5	6533.5	7481.25	5696	7687.75	3968.5	373.5	701.75	667.5	2542.25
1500000	19878.75	8080.25	7263.75	7093.75	10016.25	5563.5	938	966.25	1783	3417.75
1375000	11091.5	5558.75	5580.25	5396	9862	5309.5	1800.75	1010.75	2966.25	3542.5
1275000	6020.5	5659.25	5632	5546.5	9303.75	6151	7105.75	1632.5	5404	4972.25
1175000	2358	4741.5	5307.75	4699	8489	6445.5	26751.25	2956.75	9992.75	7233.25
1075000	649	4725.75	5268	4905	13070.5	7269.75	38836	5421.25	19672.75	10779
975000	146.75	3901.5	5767	4400.5	15223	6688.5	12475.75	8342.5	23672.75	12376
875000	27	3426	8192	3971.75	8792.25	6559.75	1589.5	12271.75	16357.5	11360.75
774000	10.75	4784.25	19763.75	5513.75	1450	9940.75	181.25	24632	10434.75	12445.5
500000	1.5	4451.25	4966.25	8436.75	16.75	14915	4.5	29268.75	830.5	10793.25
2050000	7315.5	11159	37832.75	26925.75	810.25	2385.25	153.25	1434.75	122.25	1804.75
1986000	5832.75	10080.75	5239.5	23048.5	250.75	2375	41.5	1512	51.75	2315
1923000	6407.25	2524.75	5588.75	2161.5	253	448.75	54.25	175.75	71.75	281.5
1860000	6681.25	2026.75	5355.75	1860.75	219	378.5	90	181.75	238.25	238.5
1796000	7065.75	1988.75	4992.25	2279	239.75	462.5	83.25	147.75	118.75	227
1733000	7490.25	2053.25	4450	2806.75	298.75	416.5	56.25	165.5	108.5	149.25
1670000	18184.75	5764.75	8247.5	8328.5	1292.75	1088	197.25	510.5	181.25	641
1530000	14294	4542.25	5189.75	5673.75	4108.75	1161	155	838.25	306.75	817.75
1419000	10480	3957.5	4186.75	4830.25	11297.75	1477.75	302.25	1351.5	512.5	1120.25
1308000	5740	5048.25	3843.5	4095.25	27761.75	3151.25	902.5	2704.75	1334	2363.75
1169000	1017.75	4614.75	1912.25	1964.75	15346	4866.75	2295.75	4017.5	2865	3981.25
1085000	303.75	4591.5	1424	1504.25	10481.25	6678.75	3448.75	5580.5	4664	5036
1002000	100.5	5937	1040.5	1223.25	7692.75	11175.75	5344.5	9794.25	10357.25	8989.25
919000	33.5	7496.75	704	1154.25	4903.25	17326.25	5040.5	17133.5	14756	16744.75
835000	12.25	15656.5	286.5	1615	3050.75	27933	6514.75	27975	15078.5	31623.5
774000	5.5	3572.5	90	919.5	2463.75	9130.75	65831.75	17011.25	41739.25	16175.75

Таблица 5: Среднее распределение частот по времени между регуляторами SPSA и OnDemand