Санкт-Петербургский
Государственный Университет

Математико-Механический факультет
Кафедра Системного Программирования

Божнюк Александр Сергеевич

Сравнение эффективности алгоритмов DVFS для оптимизации энергопотребления Android устройств

Отчёт по учебной практике

Научный руководитель:
ст. преп. С.Ю. Сартасов

Санкт-Петербург
2021

Содержание

Введение

На текущий момент невозможно отрицать распространенность мобильных устройств. Сейчас сложно представить себе жизнь без смартфонов, планшетов, умных часов, обладающих удобством и внушительными вычислительными мощностями. На конец 2020 года ожидается около 3.5 миллиардов [1] пользователей смартфонов. Среди мобильных операционных систем наиболее распространенным является Android [2].

Существующие мобильные устройства выполняют весьма широкий спектр задач: от установления связи с другими устройствами до проигрывания аудио и видео. В связи с этим а также с тенденцией уменьшения размеров устройств, в частности, их толщины, все сложнее обойти стороной проблему энергопотребления. От него зависит не только "время жизни" смартфона или планшета, но и его производительность.

Так как за выполнение задач смартфоном отвечает центральный процессор (ЦП, CPU), именно он является одним из узлов, требующих наибольшего количества энергии. Потребление CPU зависит от частоты и напряжения. Увеличение этих показателей, например, при установке процессора в режим максимальной производительности, приводит к быстрому истощению батареи. Однако на практике далеко не всегда процессору нужно работать именно таким образом. Зачастую имеет смысл подобрать частоту и напряжение, при которых устройство будет потреблять меньше энергии, но при этом не будет страдать пользовательский опыт. Наиболее общим подходом к уменьшению энергетических затрат при работе процессора является динамическое масштабирование частоты и напряжения (Dynamic Voltage and Frequency Scaling, DVFS), при котором энергопотребление CPU уменьшается за счет изменения рабочей частоты и напряжения.

Помимо интегрированных в операционную систему Android, существует множество других алгоритмов DVFS, не входящих в официальные сборки. При этом остается важным вопрос: какие из этих алгоритмов лучше использовать для выполнения задачи оптимизации энергопотребления? В данной работе сравнивается эффективность нескольких отобранных алгоритмов масштабирования частоты и напряжения.

1. Цели и задачи

Целью данной работы является сравнение эффективности актуальных DVFS алгоритмов. Для достижения цели были поставлены следующие задачи:

1. Провести общий обзор подсистемы CPU Frequency scaling (CPUFreq) ядра Linux, а также использованных в ней алгоритмов DVFS.
2. Провести обзор существующих алгоритмов DVFS и выделить актуальные, но не встроенные в CPUFreq.
3. Реализовать или найти реализацию нескольких алгоритмов DVFS интегрировать их в подсистему.
4. Определить метрики эффективности алгоритмов DVFS.
5. Провести сравнение эффективности реализованных алгоритмов.

2. Обзор предметной области

2.1. CPUFreq

Для управления частотами процессора в ядре Linux предусмотрена отдельная подсистема CPUFreq subsystem [3], которая состоит из 3-х частей:

1. Ядро CPUFreq — предоставляет базовую инфраструктуру и пользовательские интерфейсы для управления частотами процессора в системе. Она динамически указывает драйверам целевую специфичную частоту, на которую нужно установить процессор, используя при этом регулятор масштабирования (scaling governor).

2. Регуляторы масштабирования — модуль, реализующий алгоритмы управления частотой процессора. Регулятор может быть параметризованным.

3. Драйверы масштабирования — драйверы, которые позволяют установить процессору определенную частоту. Предоставляют регуляторам информацию о P-состояниях (парах частота-напряжение) или их диапазонах. Имеют доступ к аппаратной части, что позволяет им изменять текущие P-состояния по запросу регулятора.

Интерфейс cpufreq можно найти в подкаталоге cpufreq (например, `/sys/devices/system/cpu/cpu0/cpufreq/` для первого ядра процессора).

Файлы из подкаталога:

- `cpuinfo_min_freq`

  Данный файл показывает минимальную частоту, на которой может работать CPU (в кГц).

- `cpuinfo_max_freq`

  Данный файл показывает максимальную частоту, на которой может работать CPU (в кГц).

- `cpuinfo_transition_latency`

  Данный файл показывает время, которое необходимо процессору для переключения между двумя частотами в наносекундах. Если параметр неизвестен, будет возвращено -1.

- `scaling_driver`

  Данный файл показывает, какой драйвер cpufreq используется на текущий момент.

- `scaling_available_governors`

  Список доступных в ядре регуляторов CPUFreq.

- `scaling_governor`

  Данный файл показывает показывает активированный в текущий момент регулятор. Запись в него имени другого регулятора позволяет изменить активированный.

- `cpuinfo_cur_freq`

  Текущая частота процессора, полученная из аппаратного обеспечения (в кГц). Это частота, на которой фактически работает процессор в настоящее время.

- `scaling_available_frequencies`

  Список доступных частот (в кГц).

- `scaling_min_freq` и `scaling_max_freq`

  Данные файлы показывают текущие "границы политики"(policy limits) (в кГц). Границы можно менять, добавляя новые значения в файлы.

- `affected_cpus`

  Список работающих процессоров, которым требуется программная координация частоты.

- `related_cpus`

  Список работающих и выключенных процессоров, которым требуется программная координация частоты.

- `scaling_cur_freq`

  Текущая частота процессора, определяемая регулятором и ядром cpufreq, в кГц. Это частота, на которой ядро считает, что процессор работает.

- `bios_limit`

  Если BIOS сообщает ОС ограничить ЦП более низкими частотами, пользователь может считать максимальную доступную частоту из этого файла. Обычно это может происходить через настройки BIOS, ограничения, инициируемые служебным процессором или другими реализациями на основе BIOS или апаратуры.

- `scaling_setspeed`

  Этот файл позволяет устанавливать рабочую частоту процессора на конкретное значение в пределах между `scaling_min_freq` и

`scaling_max_freq`, если текущий регулятор является регулятором ”userspace”.

2.2. Регуляторы частоты CPUFreq

В данном разделе представлен обзор регуляторов частот ядра CPUFreq, 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 [4].

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

2. **Powersave** — держит частоту на минимально возможном уровне, что позволяет существенно увеличить время работы устройства от аккумулятора.

3. **Userspace** — позволяет пользователю или любой программе с правами суперпользователя устанавливать свою частоту ядра процессора.

4. **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 резких 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` * 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). Домно-

жение на 1000 происходит, потому что задержка измеряется в наносекундах.

- `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. В противном случае частота будет пропорциональна нагрузке.
- `ignore_nice_load (0 или 1)`: по умолчанию установлено в 0. В этом случае при расчете нагрузки все процессы учитываются, подсчет ведется по параметру cpu utilization. Если установить 1, то процессы, выполняемые с «nice» значением, учитываться не будут при расчете общей нагрузки. То есть, 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 частоты 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.

5. **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 значение (параметр `down_threshold`) 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 фактора частота будет понижаться в `sampling_down_factor` медленнее, чем повышаться.

6. **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 бысто 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:

   - Если нагрузка превысила порог `go_hispeed_load`, то частота повышается до максимальной.
   - 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 течение `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 проверятся нагрузка на процессор вне режима простоя.
- `input_boost`: Изначально равен 0. Если не 0, увеличить частоту всех ядер до `hispeed_freq` при работе с сенсорным 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.

2.3. Обзор существующих алгоритмов DVFS

Для поиска существующих алгоритмов использовался сервис "Google Scholar". Было произведено два запроса: "DVS Dynamic Voltage Scaling power consumption algorithms ОС android"и "dvfs algorithm android". По каждому из запросов были изучены первые 2 страницы выдачи поисковой системы (20 статей). Все 40 статей были рассмотрены на предмет соответствия данной предметной области в рамках системного обзора литературы [20]. В результате было отобрано только 7 статей, подходящие под наши критерии. Однако еще были рассмотрены источники, найденные в ходе совместной работы студентов Математико-Механического факультета СПБГУ Александра Данилевского и Никиты Тарасенко [21]. В конечном итоге было рассмотрено 12 решений, о которых и пойдет речь далее.

Описываемые в обзоре алгоритмы можно разделить на три специальных класса:

- Основанные на проведении определенных предварительных вычислений или получения данных. Результаты используются при выборе оптимальной частоты. К этому классу можно отнести алгоритмы [5–7, 16].

- Использующие машинное обучение. Применяются регрессионные модели (допустимый ответ на задачу - число или вектор). Важно отметить использование нейронный сетей, которые позволяют прогнозировать оптимальную частоту CPU. К этому классу относятся алгоритмы [9–13].

- Адаптивные алгоритмы, которые на основе данных о текущем состоянии (например, температуре [8] или множеству параметров [14]) вычисляют оптимальную частоту. К этому классу относятся алгоритмы [8, 14, 15].

В статье [5] описывается подход, при котором оптимальная частота выбирается в зависимости от обратной связи пользователя. Используются технологии DVFS и User Driven Frequency Scaling (UDFS). Сначала выбирается оптимальная начальная частота, и далее, в зависимости от обратной связи пользователя (получаемой при помощи UDFS), уровень частоты постепенно уменьшается до того момента, пока пользователь не почуствует дискомфорт. В итоге имеем преимущество в 25% по сравнению со стандартными DVFS и в 3% по сравнению с существующей UDFS.

В [6] алгоритм основан на введении такого понятия, как операционная интенсивность (Operational intensity, OI). OI вычисляется по формуле (Number of operations) / (BUS access * Byte). Составлена таблица соответствия оптимальных частот значениям OI. Алгоритм заключается в подсчете OI и выборе оптимальной частоты согласно таблице. В итоге

достигнута экономия энергии 9% по сравнению с Performance и 8% по сравнению с Ondemand.

В работе [7] представлен так называемый Usage-History DVFS (UH-DVFS) алгоритм. Идея состоит в измерении вычислительных требований к часто вызываемым сценариям использования мобильного устройства и записи их в специальную структуру данных, называемую таблицей пользовательских транзакций (User Transaction Table, UTT). Каждый раз, когда запущенная пользователем транзакция обнаруживается, происходит поиск в таблице. При совпадении записанное вычислительное требование будет использовано для установления требуемой частоты. Приводится результат: экономия энергии от 10% до 36%.

В [8] описывается подход, в котором учитывается температура процессора при определении частоты. Сначала измеряется температура CPU. Если она меньше определенного порога, то выставляется частота в зависимости от нагрузки на процессор. Если нагрузка больше 80%, DVFS устанавливает максимальную частоту. Когда есть более низкую частота, которая держит нагрузку ниже порога использования, устанавливается именно она. Если температура выше определенного порога, запускается новый алгоритм, который устанавливает целевую частоту в зависимости от выбранной опции оптимизации (power / performance) и удерживает её. Также определены специальные случаи, при которых программа возвращается обратно к установке частоты согласно нагрузке. Демонстрируются следующие результаты: при использовании опции power энергопотребление уменьшается в среднем 12.7%. Производительность (время исполнения) повышается в среднем на 6.3% при опции performance, энергопотребление снижается на 6.7%.

Нельзя обойти стороной алгоритмы, основанные на машинном обучении. Так, в статье [9] описан метод, который при помощи так называемых Встречных Систем Распространения (Counter Propagation Networks, CPN) классифицирует поведения задач и прогнозирует наилучшую частоту или напряжение для системы. На вход CPN подаются производительность (вычисляется, от 0 до 100%), количество исполненных инструкций, промахи кэша данных и инструкций. На выходе - целевая частота. Данные для тренировки нейросети берутся из различных бенчмарков, приведенных в самой статье. Подход разработан как для одноядерной, так и многоядерной системы. Приводятся следующие результаты: на одноядерной системе: экономия от 5% до 20% энергии с ограничением потери производительности (ПП) 10% и с 9% до 42% экономии энергии с ограничением потери производительности 30%. В многоядерных системах наблюдается экономия от 2.3% до 8.8% энергии с ограничением потери производительности 10% и с 3.9% до 22% энергии с ограничением потери производительности 30%.

В [10] можно увидеть еще один подход, связанный с машинный обучением. Для подбора оптимальной частоты прогнозируется нагрузка на процессор. Метод реализован через систему управления питанием Long

Short-Term Memory (LSTM). Система изучает шаблоны использования процессора в различных случаях и прогнозирует нагрузку на процессор в текущей ситуации. Пока нейронная сеть не натренирована, используется Interactive регулятор. В итоге получаются следующие результаты: снижение энергопотребления мобильных процессоров максимум на 19% по сравнению с существующей системой управления питанием Android.

Основная идея алгоритма в [11] состоит в том, программист собственноручно разбивает свою программу на блоки (фреймы) при помощи специальных аннотаций в программном коде, в которых указывает тип нагрузки и желаемая производительность. Вся эта информация подается на вход регулятору, который состоит из двух частей: модуль предсказания рабочей нагрузки и модуль определения оптимального напряжения и частоты. Первый модуль по данным о фрейме считает нагрузку на систему при исполнении этого фрейма и такой показатель, как deadline, при помощи метода Exponential Weighted Moving Average (EWMA). Второй модуль принимает предсказания нагрузки от нескольких фреймов и по ним определяет оптимальные частоту и напряжение при помощи Q-Learning алгоритма. Имеется следующий результат: минимизация потребления энергии до 30% по сравнению с существующими регуляторами.

В статье [12] описан подход, где ожидаемое использование процессора предсказывается при помощи модификации EWMA: AEWMA-MSE. Далее используется Q-Learning алгоритм, определяющий по предсказанной нагрузке, в какое состояние должна перейти система. Приведенный результат: снижение потребления на 29% по сравнению с существующими в системе подходами.

В [13] используется две технологии: DVFS регулятор и Dynamic Core Selection (DCS). Для оптимизации алгоритм использует Q-Learning. На вход подаются температура и суммарное количество циклов ядер процессора (состояний) с момента последней итерации алгоритма. Далее подсчитывается специальная payOff функция, на основе которой обновляются значения в так называемой Q-Table - таблице, сопоставляющей состоянию и конфигурации системы (частота и включенные ядра) новую конфигурацию, в которую нужно перейти системе. При помощи EWMA предсказывается следующее состояние системы (количество циклов ядер с текущего момента до следующего вызова алгоритма). Далее на основе предсказанного состояния при помощи Q-Table выбирается конфигурация, на которую должна перейти система. Приводится следующий результат: в среднем улучшение на 22% (7-40%) по сравнению с существующими методами.

В статье [14] представлена платформа, который поддерживает как DVFS, так и DPM (Dynamic Power Management), отслеживает загрузку процессора приложениями и адаптивно отслеживает включает / выключает ядра или изменяет частоту, чтобы снизить потребление. Система состоит из трех компонентов: Online Resource Usage Monitor, который

собирает информацию о состоянии каждого ядра системы (частота, загруженность, температура и прочие параметры), Policy Manager, позволяющий пользователю настраивать систему, например, выставлять ограничения на диапазон частот, а также Low-Power Controller, занимающийся изменением частоты, напряжения и включением / отключением ядер. Алгоритм основан на вызове одной из двух функций, отвечающих за уменьшение и увеличение производительности системы. При вызове они либо отдают приказ регулятору о нагрузке, либо об отключении мощного ядра. Решение о вызове функции принимается на основе значения скользящего среднего по загруженности процессора. В итоге имеется следующий результат: снижение энергопотребления от 22% до 79%, при этом незначительно влияя на производительность.

В [15] описан регулятор, построенный на базе классического Ondemand, дополненного возможностью включать и отключать (переводить в спящее состояние) неиспользуемые ядра. Изначально выбран порог для отключение ядра процессора, равный 10%. Если нагрузка на ядро меньше этого порога, ядро переходит в спящий режим. Также представлена формула, на основе которой можно высчитать новую частоту процессора, частоту, которую предлагает установить Ondemand, общую нагрузку, количество активных ядер и общее количество ядер. Также сформулирован алгоритм регулировки так называемой пропускной способности ядра в зависимости от его загруженности. В итоге имеется следующая последовательность действий: запуск Ondemand, изменение пропускных способностей ядер, проверка того, нужно ли отключить какие-либо ядра, отключение, обновление частот. Как результат: экономия до 14% энергии при динамических нагрузках и до 22% при более постоянной нагрузке в сравнении с базовыми регуляторами в Android.

Решение, описанное в [16], не решает проблему в общем случае, а позволяет оптимизировать лишь конкретное приложение под конкретную платформу. Интересно то, что управление частотой происходит через базовый регулятор Userspace, а не реализован новый. Процесс оптимизации разбит на две стадии. Первая стадия - offline-профилирование, при котором оптимизируемое приложение запускается на нескольких комбинациях пар частоты и скорости памяти, и при каждой конфигурации замеряются различные ключевые метрики (потребленная мощность, увеличение производительности относительно самой слабой комбинации). Полученные значения интерполируются по всем возможным комбинациям частоты и скорости памяти. На этом этапе также происходит запуск приложения с базовым регулятором и оценивается его производительность - R. Вторая стадия - online controlling, при которой в каждый момент времени оценивается разница между текущей производительностью и желаемой производительностью R. Происходит оценка, при которой определяется, как их текущего состояния попасть в состояние R. Выбирается набор переходных состояний таким образом, чтобы переход по ним был оптимален с точки зрения производительности и затрачен-

ного времени. Такая оптимизация была проведена с 6-ю популярными по их мнению приложениями, имеются следующие результаты (приведены в виде пар: выигрыш по производительности / выигрыш по энергопотреблению): -0.4%/25.3%, 4.1%/15.3%, 0.6%/14.9%, -0.4%/27.2%, 0.0%/4.2%, 9.3%/31.6%

Таблица 1 содержит в себе описанные выше алгоритмы. К тому же, в ней описаны устройства, на которых тестировались приведенные в обзоре решения.

Таблица 1: Обзор алгоритмов DVFS

Статья (авторы)	**Класс алгоритма**	**Достигнутые результаты**	**Тестовый стенд**
Akter Rumi, Asaduzzaman and Hasibul Hasan [5]	Предварительные вычисления	Экономия 25% по сравнению со стандартными DVFS Экономия 3% по сравнению с существующей UDFS	Samsung Galaxy SII (ARM Cortex-A9 1.2 ГГц)
S.J. Cho, S.H. Yun and J.W. Jeon [6]	Предварительные вычисления	Экономия 9% по сравнению с Performance Экономия 8% по сравнению с Ondemand	Встроенная система (ARM Cortex-A15 1.7 ГГц) Android 4.1 м
X. Li, W. Wen X. Wang [7]	Предварительные вычисления	Экономия 10% - 36%. Влияния на качество работы пользователей нет	Xiaomi-2S (Snapdragon APQ8064T 1,7 ГГц) Android 4.1
K. Poornambigai, M. L. Raj and P. Meena [8]	Адаптивный алгоритм	Опция power: экономия 12.7%. Опция performance: экономия 6.3%	Odroid-Q (Exynos 4412 1,4 ГГц) Android 4.0
Y. L. Chen M. F. Chang et al. [9]	Машинное обучение	5% - 20% (одноядерный, ПП* - 10%) 9% - 42% (одноядерный, ПП - 30%) 2.3% - 8.8% (мультиядерный, ПП - 10%) 3.9% - 22% (мультиядерный - 30%) *ПП - ограничение потери производительности	NVIDIA JETSON (ARM Cortex-A15 2.3 ГГц)
J. Lee, S. Nam, and S. Park [10]	Машинное обучение	Экономия 19% по сравнению со стандартными DVFS	Google Nexus 6P (Snapdragon 810 2.0 ГГц)
Luis Alfonso Maeda-Nunez [11]	Машинное обучение	Экономия до 30% по сравнению со стандартными DVFS	BeagleBoard-xM с DM3730 SoC (ARM Cortex A8 600 МГц - 1 ГГц)
S. A. Carvalho, D. C. Cunha and A. G. Silva-Filho [12]	Машинное обучение	Экономия 29% по сравнению со стандартными DVFS	Motorola XT1033 (ARM Cortex A7) Android 4.4
A. Das, M. J. Walker et al. [13]	Машинное обучение	Экономия 22% (7-40%) по сравнению с существующими методами	NVIDIA JETSON (ARM Cortex-A15 2.3 ГГц)
J. Choi, B. Jung et al. [14]	Адаптивный алгоритм	Экономия 22% - 79%. Влияние на производительность на всех стендах незначительное	Odroid-XU3 (Samsung Exynos 5422 SoC) Raspberry Pi2 (ARM Cortex-A7 900 МГц) Pine 64 (ARM Cortex A53 1.2 ГГц)
L. Broyde, K. Nixon et al. [15]	Адаптивный алгоритм	Экономия до 14% при динамических нагрузках Экономия до 22% при постоянных нагрузках	Nexus 5 (Snapdragon 800 2.3 ГГц) Android 6.0
Karthik Rao, Sudhakar Yalamanchili, Yorai Wardi [16]	Предварительные вычисления	Результаты по 6 приложениям (пары выигрыш по производительности / по энергопотреблению) -0.4%/25.3%, 4.1%/15.3% 0.6%/14.9%, -0.4%/27.2%, 0.0%/4.2%, 9.3%/31.6%	Nexus 6 (Snapdragon 805 2.7 ГГц) Android 6.0

2.4. Обзор технических средств

2.4.1. Выбор тестового стенда

В качестве тестового стенда для экспериментов был выбран смартфон под названием Xiaomi Redmi Note 8 Pro, имеющий следующие характеристики:

- Оперативная память: 6 Гб;
- Основная память: 64 Гб;
- Операционная система: Android 9 с оболочкой MIUI-11. Впоследствии была обновлена до Android 10;
- Процессор: Mediatek Helio G90T, имеющий 8 ядер: 2 высокопроизводительных ядра архитектуры ARM Cortex-A 76 и 6 менее мощных, но более энероэффективных ядер ARM Cortex-A55.

Данный выбор обусловлен высокой популярностью смартфона в России [17], наличием большого количества материала по работе с ним. Также это мобильное устройство поддерживает последние версии операционной системы Android, что способствует большей актуальности текущей работы, имеет современный многоядерный процессор и приемлемую цену, а значит большую доступность.

2.4.2. Подготовка устройства

После выбора смартфона его следовало подготовить к тестированию. Важно отметить, что в большей степени смартфон был подготовлен студентом 4 курса Математико-Механического факультета Богдановым Евгением Алексеевичем в рамках его выпускной квалификационной работы. Ниже описаны шаги, которые были проделаны Евгением.

Для установки новых прошивок и регуляторов на выбранный смартфон была произведена установка прав суперпользователя. После было выбрано и собрано ядро из проекта с открытым исходным кодом AgentFabulous begonia kernel project [22]. Далее была установлена прошивка POSP - Potato open source project [23]. Она наиболее близка к проекту AOSP [24] - реализации Android в его чистом виде, но разработана непосредственно для используемого нами устройства. К тому же, ядро разрабатывается теми же авторами, что обеспечивает наилучшую совместимость.

2.4.3. Установка прошивки

Для установки POSP были проделаны следующие шаги:

1. Скачан образ прошивки POSP, собран в нужной конфигурации, распакован и загружен на microSD карту.

2. При помощи программы для восстановления Android устройств Team Win Recovery Project (TWRP [26]) было осуществлено форматирование разделов Dalvik Cache, Data и Cache, а также проведено затирание (wipe) памяти всего телефона.

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

4. Заново получены права суперпользователя на новой прошивке при помощи установки системного приложения magisk [27].

2.4.4. Установка регулятора DVFS

Неотъемлимой частью данной работы является добавление новых регуляторов в систему. Это требуется для возможности проведения тестовых испытаний. Далее будут описаны требуемые действия, которые следует произвести [18, 19].

В первую очередь важно отметить, что регулятор представляет из себя программу на языке Си, а значит в нашем распоряжении находится файл с расширением .c. Обычно имеет название типа "cpufreq_govname.c". Пусть наш файл будет называться "cpufreq_mygovernor.c". Далее нужно:

1. Поместить файл с регулятором в каталог `/drivers/cpufreq`, описанный ранее.

2. Модифицировать файл `kconfig`, добавив в него выбор нового регулятора, чтобы тот отображался в нашей конфигурации системы:

```
config CPU_FREQ_GOV_MYGOVERNOR
tristate ‘‘mygovernor’ cpufreq governor"
depends on CPU_FREQ
help
’mygovernor’ - my custom governor!
```

3. Снова модифицировать `kconfig` файл, добавив в него следующий код, объявляющий возможность того, что добавляемый регулятор будет выбран в качестве регулятора по умолчанию:

```
config CPU_FREQ_DEFAULT_GOV_MYGOVERNOR
bool ‘‘mygovernor’’
select CPU_FREQ_GOV_MYGOVERNOR
help
Use the CPUFreq governor ‘mygovernor’ as default.
```

4. Находясь в том же каталоге `/drivers/cpufreq`, модифицировать `Makefile`, добавить в него следующую строку:
`obj-$(CONFIG_CPU_FREQ_GOV_MYGOVERNOR) += cpufreq_mygovernor.o`

5. Переместиться в каталог `kernel_source/includes/linux`, открыть файл `cpufreq.h`. Найти следующий макрос, описывающий регулятор как структуру (пример для Ondemand):

   ```
   #elif defined(CONFIG_CPU_FREQ_DEFAULT_GOV_ONDEMAND)
   extern struct cpufreq_governor cpufreq_gov_ondemand;
   #define CPUFREQ_DEFAULT_GOVERNOR (amp;cpufreq_gov_ondemand)
   ```

   Здесь перечислены и другие доступные регуляторы. Наша задача - добавить сюда макрос со своим регулятором:

   ```
   #elif defined(CONFIG_CPU_FREQ_DEFAULT_GOV_MYGOVERNOR)
   extern struct cpufreq_governor cpufreq_gov_mygovernor;
   #define CPUFREQ_DEFAULT_GOVERNOR (amp;cpufreq_gov_mygovernor)
   ```

6. При верном выполнении всех шагов мы можем выбрать добавленный регулятор в меню (`menuconfig, xconfig, gconfig, nconfig`). При выборе в меню он сразу добавляется в ядро. Существует возможность смены регулятора: для этого нужно заменить значение в файле `cpufreq/scaling_governor`.

3. Ход работы

3.1. Выбор метрик сравнения

Для проведения сравнения требовалось выбрать метрики, по которым в конечном итоге будет проводиться сравнения. По результатам анализа методологий проведения экспериментов [28] из главы 2.3. были найдены следующие метрики для энергопотребления и производительности:

Для анализа потребления энергии:

- Разница заряда батареи (в %);
- Потребляемая мощность (в ваттах, милливаттах);
- Потребляемая энергия (в джоулях, миллиджоулях, миллиампер-часы).

Для анализа производительности:

- Количество кадров в секунду или FPS;
- Количество условных единиц (баллов).

Основная проблема такой метрики, как разница заряда батареи заключается в трудности её получения с достаточной точностью. Похожая проблема у мощности: для получения данных приходится прибегать к внешнему воздействию (например, подключать приборы для измерения силы тока и напряжения). Однако потребляемую энергию можно подсчитать. Для этого можно получить время проведения процессора на конкретных частотах (см. главу 3.3.4.).

Количество кадров в секунду (FPS) является достаточно хорошей метрикой для оценки производительности. Однако её получение в нашем случае сопряжено с многочисленными трудностями. Так, Adreno Profiler [29] не способен работать с процессором Mediatek. К тому же, обычные приложения для измерения FPS могут только выводить значение в текущий момент времени, однако получить записанные значения для последующего анализа не представляется возможным. Командные утилиты способны выводить время отрисовки конкретного кадра, но не сам FPS. А такие программы, как GameBench [30], которые позволяют вывести график FPS за определенный промежуток времени, требуют корпоративного доступа. Однако для оценки производительности смартфона довольно популярным подходом является использовать различные бенчмарки. В них производительность принято оценивать в условных единицах (баллах). Эта метрика является интегральной метрикой общей производительности. Она не дает возможности провести детальную оценку, однако позволяет получить общее впечатление от работы смартфона. Подробнее об этом также рассказано в главе 3.3.4.

3.2. Выбор алгоритмов для тестирования

Самой существенной частью работы является подбор алгоритмов для тестирования. Авторы работ, рассмотренных в главе 2.3. не предоставили исходные коды регуляторов, а реализовывать эти весьма не тривиальные алгоритмы самостоятельно не представляется возможным из-за ограниченности по времени.

Было принято решение искать готовые регуляторы. В интернете можно найти большое количество ядер Linux, в которых есть различные пользовательские алгоритмы, использующиеся людьми [31]. В начале эти ядра были найдены, а алгоритмы - собраны [32]. Однако при установке регуляторов в текущее ядро пришлось столкнуться с действительно большим количеством трудностей. Так, все регуляторы были взяты из модифицированных ядер, а потому они требовали разрешения большого количества различных зависимостей, что далеко не всегда представлялось возможным и приводило к ошибкам при сборке ядра. А потому большинство алгоритмов из отобранных пришлось отбросить за неимением возможности их установки в текущее ядро. Несмотря на многочисленные проблемы, разрешить зависимости в нескольких случаях удалось, что привело к успешной установке девяти дополнительных регуляторов. Ниже приведен их список.

1. **Schedutil** [33] — регулятор, который для выбора следующей частоты использует информацию от планировщика задач по использованию CPU. Частота высчитывается по довольно простой формуле. Регулятор предоставляет так называемые обработчики обновлений утилизации, которые и выполняют необходимые вычисления. Также этот регулятор поддерживает быстрое переключение частоты, если это поддерживается текущим драйвером cpufreq. Если это самое быстрое переключение используется, то операции регулятора выполняются в обработчиках обновления утилизации. В противном случае операции переключения частоты выполняются отдельного рабочего элемента (work item), вызывающий метод для обновления частоты до значения, уже вычисленного заранее в одном из обработчиков обновлений утилизации. Регулятор совмещает в себе опции настройки Ondemand и Conservative. Далее идут разного рода модификации данного регулятора.

2. **Alucardsched** — регулятор, который позиционируется как модификация Schedutil и обещает хороший баланс между производительностью и потреблением батареи;

3. **Blu_schedutil** — модификация Schedutil, которая обещает более лучшую экономию батареи по сравнению с Schedutil;

4. **Lenovoutil**;

5. **Lenutil**;
6. **Nubiautil**;
7. **Realmeutil**;
8. **Renoutil**;
9. **Schedutil_pixel**.

Как можно заметить, про большую часть регуляторов нет достоверной информации. Однако можно увидеть, что они все в той или иной степени являются модицфикациями Schedutil, взятых из ядер мобильных устройств. Данное обстоятельство не влияет на репрезентативность работы, поскольку она как раз и ставит перед собой задачи исследования поведения этих регуляторов и сравнения их друг с другом.

3.3. Эксперименты

Существенной частью работы является проверка найденных алгоритмов на эффективность. Для этого были определены инструменты, с помощью которых можно проводить тесты, выбран набор тестовых случаев, методология проведения тестирования. Далее были произведены тестирование и анализ полученных результатов.

3.3.1. Выбор инструмента для тестирования

Для автоматизации тестирования требовалось выбрать инструмент, которое позволяло бы написать и запустить тесты, эмулирующие определенную работу с GUI на телефоне. Следует выделить требования, которым данный инструмент должен удовлетворять:

- Отсутствие требования доступа к исходному коду приложения;
- Возможность взаимодействовать с приложением в случае, если его верстка выполнена не средствами Android;
- Доступ к Android Debug Bridge (ADB) - инструменту для взаимодействия с устройством при помощи командной строки [34].

Было рассмотрено определенное количество возможных вариантов подобных инструментов. Например, Espresso [35] обладает большим количеством преимуществ, что делает его текущим стандартом для тестирования графического интерфейса в индустрии, однако данная платформа требует доступа к исходному коду приложения, что не позволяет использовать её для исследования. UiAutomator [36] уже не требует доступ к исходному коду, однако не имеет возможности работать с верстской, выполненной не с помощью средств Android (например, игра на Unity). К тому же, UiAutomator не предоставляет возможность использования ADB. Robotium [37] и Selendroid [38] обладают теми же свойствами, что и UiAutomator, однако являются более устаревшими.

В конечном итоге выбор пал на официальную утилиту от компании Google, которая поставляется вместе с Android SDK - Monkeyrunner [39]. Она предоставляет API для написания программ на языке Python. Эти программы позволяют управлять устройством Android при помощи компьютера, для чего используется ADB. С помощью такого подхода можно совершать все необходимые действия: имитировать касания экрана в определенной точке, запускать или удалять приложения без требования к их исходному коду, получать нужную информацию при помощи ADB.

3.3.2. Тестовые случаи

Для проведения тестирования был выделен ряд тестовых случаев - возможных сценариев использования смартфона. Основные тестовые

случаи выбирались на основе работ из главы 2.3. Часть из них имитирует сложные вычислительные задачи, например, игры или съемка видео, которые выражаются в высокой нагрузке на процессор, а часть позволяет оценить поведение алгоритмов на более простых задачах, таких как просмотр видео или набор текстовых сообщений.

В конечном итоге имеем следующие тесты:

- **Camera test:** Установка приложения для съемки Open Camera[1], если оно не было установлено до этого, запуск камеры смартфона, съемка видео в течение 15 минут;
- **Typing test:** Установка приложения для заметок Notes[2], если оно не было установлено до этого, создание новой заметки и произведение набора текста в течение 15 минут в открытую заметку;
- **Flappy Bird test:** Установка игры Flappy Bird [40], если она не была установлена, запуск и имитация игровой деятельности при помощи касания экрана смартфона в течение 15 минут;
- **Trial Xtreme test:** Аналог предыдущего теста, однако запускается более требовательная к производительности смартфона игра - Trial Xtreme 3 [42];
- **Video test:** Установка приложения VLC player for Android[3] и воспроизведение 15-минутного видео.

Указанные тесты были реализованы на языке Python и были выложены в качестве открытого репозитория на платформу GitHub[4]. Стоит отметить, что эта часть работы легла в основу статьи «On Application of Simultaneous Perturbation Stochastic Approximation for Dynamic Voltage-Frequency Scaling in Android OS», принятой на International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP) 2021 [41].

3.3.3. Подготовка смартфона

Для обеспечения наибольшей объективности результата в ходе измерения проводились следующие действия:

- Удаление всех сторонних приложений, не относящихся к проводимым тестам;
- Включение авиарежима. Это влекло за собой отключение Wi-Fi, GPS, Bluetooth и других периферийных устройств;

[1]Open Camera;
[2]Notes;
[3]VLC player for Android;
[4]GitHub репозиторий с тестами.

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: Константы энергопотребления Xiaomi Redmi Note 8 Pro

- Выдерживание между тестами десятиминутной паузы с целью корректного завершения всех фоновых процессов и остывания устройства до исходной температуры.

3.3.4. Оценка потребляемой энергии

Из подсистемы CPUFreq можно получить информацию о времени, проведенном кластером процессора на конкретной частоте за все время использования устройства, а также механизмы сброса статистики.

Тестирование проводилось следующим образом: перед каждым тестом производился сброс статистики использования частот, после завершения теста производился сбор этой самой статистики и с сохранением её на компьютер. Каждый из описанных тестов был проведен по 5 раз на каждом алгоритме.

Для подсчета потребляемой энергии потребовалось извлечь файл *power_ profile.xml*. Данный файл предоставляется производителем устройства и содержит информацию об энергопотреблении смартфона. В частности, в нем описаны константы энергопотреления кластеров процессора, при их работе на той или иной частоте в течение десятимиллисекундного интервала. Эти значения применялись в работе для подсчета энергии. Они приведены в таблице 2.

Для оценки итогового энергопотребления в миллиамперчасах была

использована следующая формула:

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

где $time\_on\_freq(i)$ — время проведенное на частоте i, полученное в результате экспериментов, $consumption\_const(i)$ — энергопотребление i-й частоты за 10-ти миллисекундный интервал из таблицы 2. Для сравнения бралось среднее значение по пяти тестам.

3.3.5. Оценка производительности

Важной частью работы также является оценка производительности смартфона при его использовании с целью проверки отсутствия отрицательных эффектов на взаимодействие пользователя с устройством.

Для оценки производительностью использовался один из наиболее популярных методов тестирования - AnTuTu benchmark[1]. Эта утилита поддерживает тестирование устройств с процессорами Mediatek, в отличие от других аналогов, что стало ключевым фактором ее выбора. Она работает следующим образом: производится ряд измерений, по результату которых устройству выставляются баллы по пяти различным категориям: CPU, GPU, Memory, UX и Total. Сам набор тестов включает в себя тестирование обработки графики, различные вычисления для тестирования CPU с целыми числами и числами с плавающей запятой, тесты скорости работы оперативной памяти, тесты работы с обычными приложениями (например, перемотка в социальных сетях). По информации о баллах можно делать определенные выводы о том, как применение различных регуляторов влияет на поведение устройства.

[1] AnTuTu benchmark.

3.4. Анализ результатов

После проведения тестирования результаты были вынесены в таблицу Excel, где уже были проведены главные расчеты[1]. По результатам вычислений были построены диаграммы, которые будут показаны ниже. Важно отметить, что настройки каждого регулятора никак не изменялись. То есть, каждый регулятор тестировался в режиме настроек "по умолчанию".

3.4.1. Энергопотребление

Финальные результаты тестирования энергопотребления представлены на диаграммах ниже. В них каждому столбцу соответствует отдельный регулятор. Высота столбца - количество потребленной энергии в миллиампер-часах (мАч). Точные значения энергопотребления можно найти в таблице с результатами вычислений.

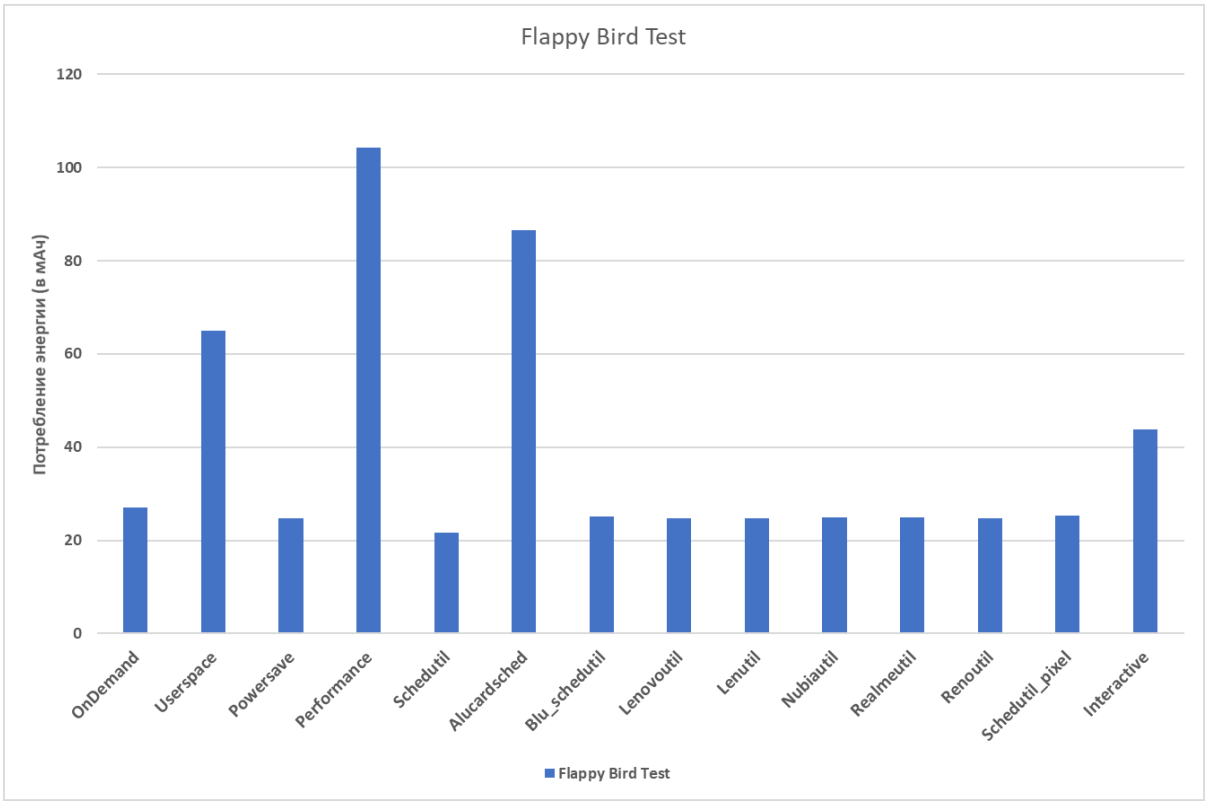

Рис. 1: Тест: игра Flappy Bird

Рассмотрим рисунок 1, показывающий, какое примерное количество энергии было потреблено каждым регулятором при игре в Flappy Bird. Сразу заметно, что новый регулятор Alucardsched показывает большое значение потребления энергии. Если посмотреть результаты вычислений

[1]Результаты вычислений.

в таблице, то можно заметить, что на первом кластере он выставляет частоту всегда на максимальное значение, а на втором кластере - на минимальное. Таким образом, в данном тесте он находится на втором месте по потреблению энергии, уступая Performance, который на обоих кластерах устанавливает максимальную частоту. Большое потребление показывает и регулятор Userspace с настройками по умолчанию.

Если смотреть на остальные новые регуляторы, то их значения потребления в целом похожи. Однако интересно то, что модификации регулятора Schedutil показывают немногим большие значения потребления, чем сам регулятор Schedutil. Более того, Schedutil показывает даже большую экономию, чем регулятор Powersave. Если посмотреть на таблицу, то можно увидеть, что Powersave выставлял минимальную частоту только на втором кластере, но не на первом. Если говорить о регуляторе Interactive, то он показывает большие значения потребления, чем Ondemand, однако все же является более экономным, чем Userspace.

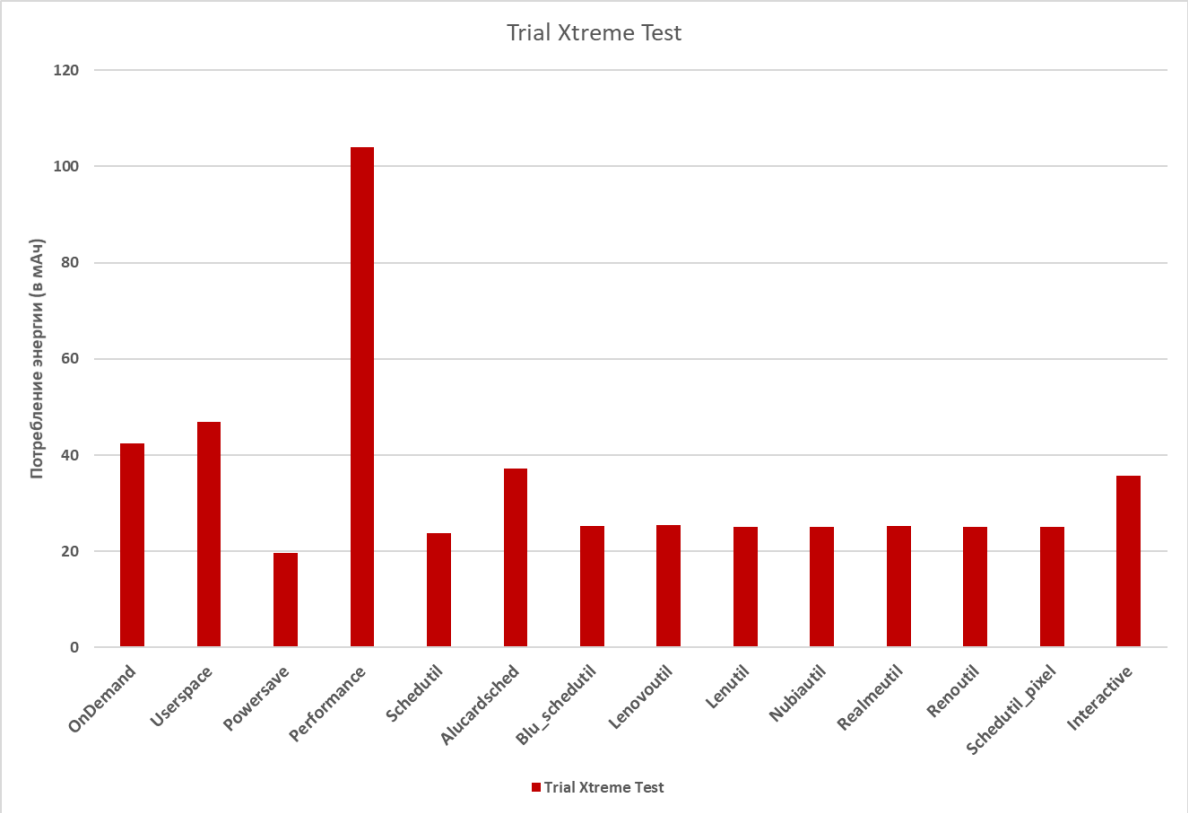

Рис. 2: Тест: игра Trial Xtreme

Теперь рассмотрим рисунок 2, где показаны значения потребления энергии при игре в Trial Xtreme 3.

Здесь можно сразу заметить, что Alucardsched показывает уже большую экономию, даже по сравнению с Userspace и Ondemand. Userspace и Ondemand на этот раз показывают большие значения потребления энер-

гии, нежели Interactive. Это можно назвать отличием по сравнению с результатами на рисунке 1.

Если смотреть на Schedutil и другие его модификации (помимо Alucardsched), то их значения энергопотребления находятся примерно на одном уровне, причем их поведение не сильно отличается от поведения в предыдущем тесте. Однако все же они потребляют энергии больше, чем Powersave.

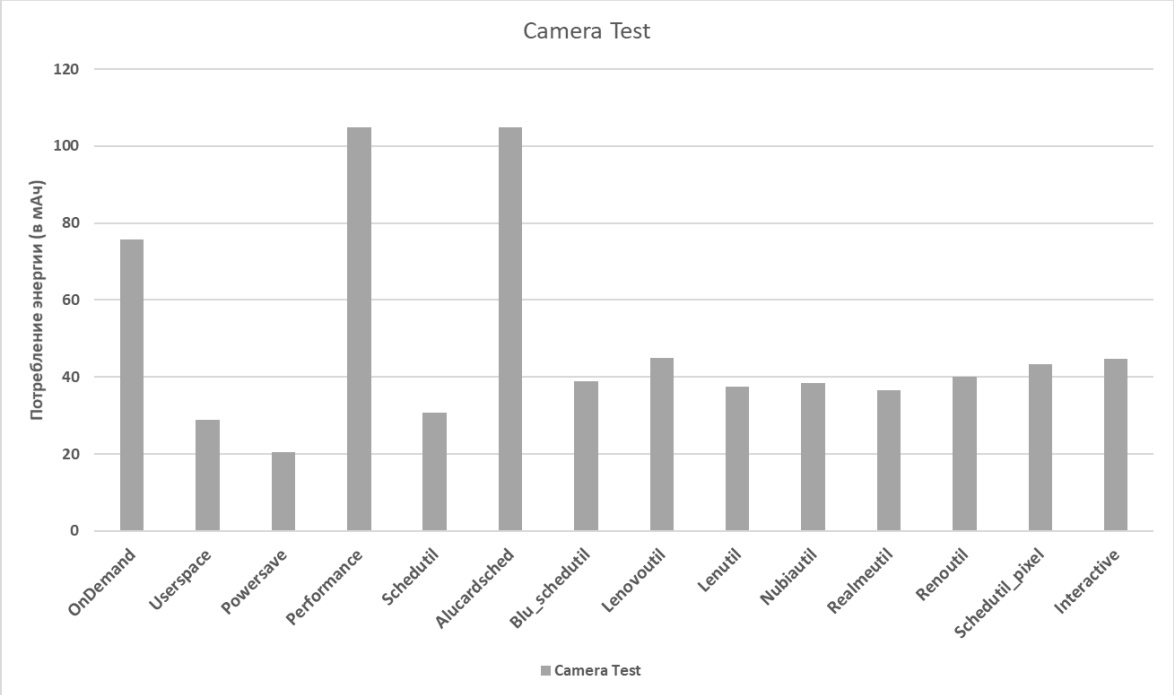

Рис. 3: Тест: Съёмка видео

Далее можно рассмотреть рисунок 3, где показаны результаты результаты теста по съемке видео.

Самые высокое потребление энергии показывают регуляторы Performance и Alucardsched. При этом очень большие значения и у Ondemand. Самый экономный регулятор на этот раз - Powersave. Немного больше результаты у Userspace, однако он не превосходит Ondemand.

Если смотреть на новые регуляторы (помимо Alucardsched), то тут можно увидеть, что сам Schedutil на этот раз потребляет энергии примерно столько же, сколько и Ondemand. Однако остальные новые регуляторы оказались более энергопотребляющими. Так, Lenovoutil и Schedutil_pixel превысили планку в 40 мАч, чего до этого не происходило. Эти результаты также выше, чем у Lenutil, Nubiautil и других, однако сравнимы с результатом Interactive.

Отдельно имеет смысл отметить, что в этом тесте модификации Schedutil (исключая Alucardsched) уже не показывают примерно одинаковый результат энергопотребления.

Далее рассмотрим рисунок 4, где демонстрируются результаты теста по воспроизведению видео.

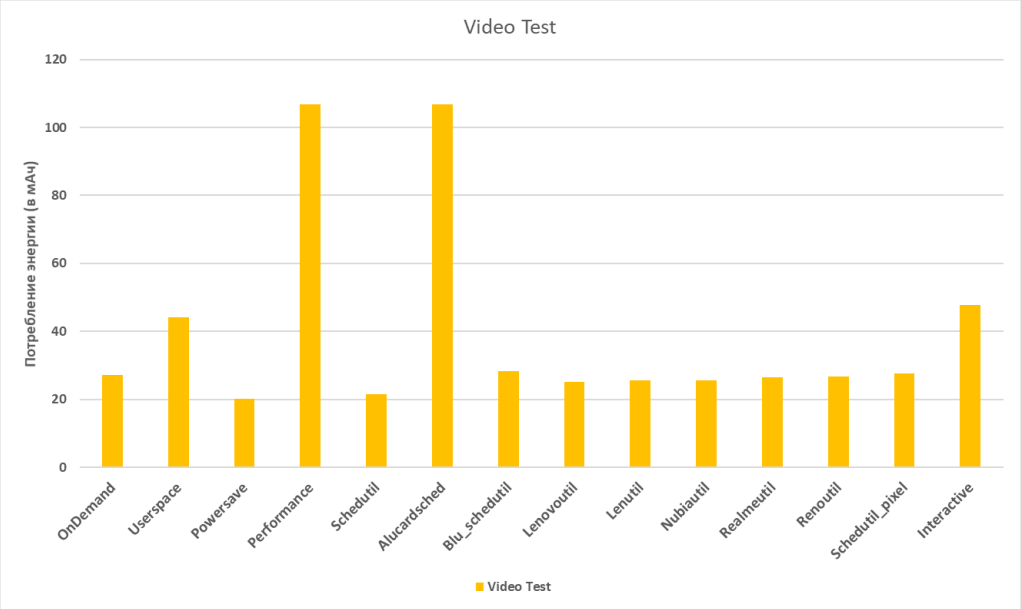

Рис. 4: Тест: Воспроизведение видео

Как и в двух предыдущих случаях из трех, Alucardsched по значению потребления очень близок к Performance. Другие модификации Schedutil все так же превышают сам Schedutil по количеству потребленной энергии и тоже примерно равны между собой. В общем и целом, ситуация напоминает случай рисунком 1 (отличие разве что в поведении Alucardsched и Schedutil).

Наконец, можно рассмотреть результаты тестирования набора текста, продемонстрированные на рисунке 5.

Необычно здесь то, что показатели у модифицированных регуляторов (опять таки, исключая Alucardshed) значительно выше, чем в предыдущих случаях. Blu_schedutil, Lenovoutil, Lenutil, Nubiautil, Realmeutil, Renoutil, Schedutil_pixel по меркам потребления энергии почти достигли планки в 60 мАч, чего в предыдущих тестах не наблюдалось. Также довольно высокие показатели и у Interactive, и у Ondemand, и у Userspace. Schedutil также повысил свои показатели, однако не столь значительно. Довольно занимательным является тот факт, что множество регуляторов резко увеличили количество потребленной энергии на, казалось бы, весьма простой нагрузке.

Если сравнивать регуляторы между собой, то в общем и целом, ситуация также похожа на предыдущие тесты. Alucardsched и Performance показывают наибольшее потребление. Наибольшая экономия демонстрируется регулятором Performance, при этом Schedutil, как и в прошлые разы, не очень сильно от него отстает. Модификации регулятора Schedutil потребляют примерно одинаковое количество энергии, но при этом большее, чем сам Schedutil. Регулятор Interactive на данном типе нагрузки показал большее энергопотребление, чем и Ondemand, и новые добавлен-

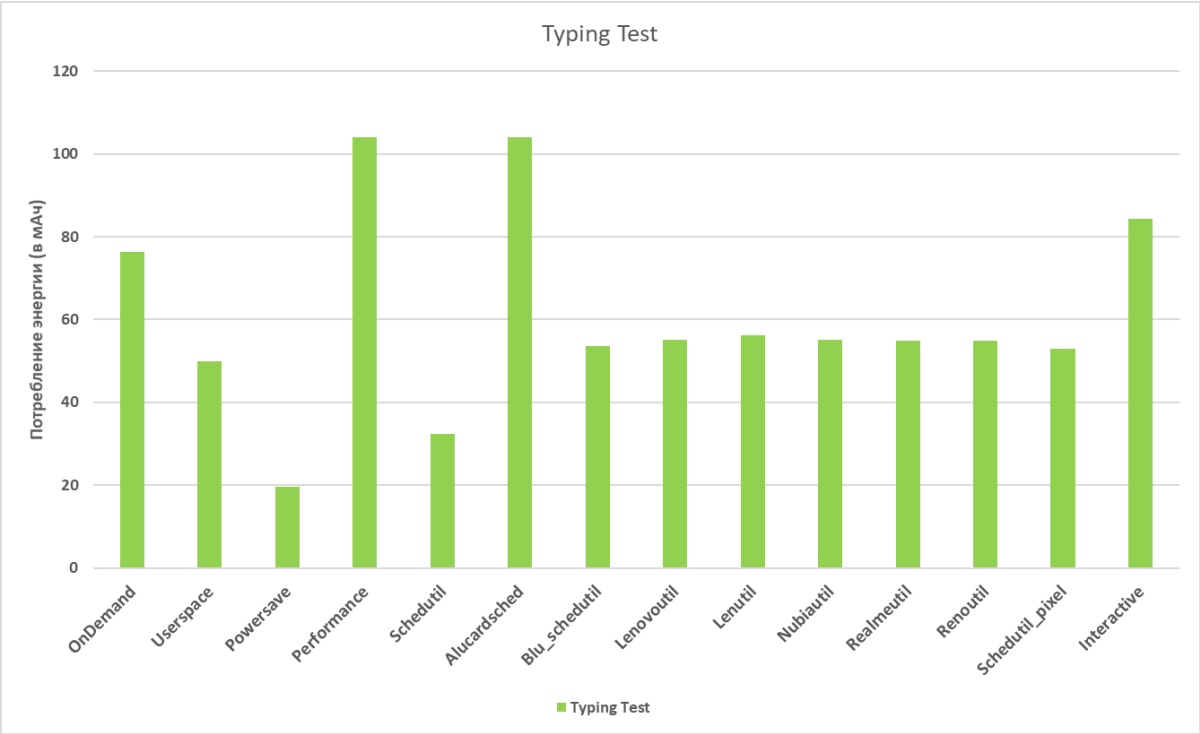

Рис. 5: Тест: Печать на клавиатуре

ные регуляторы, но все же меньшее, чем у Performance с Alucardsched.

3.4.2. Производительность

На диаграмме ниже представлены результаты тестов производительности. Каждая группа столбцов состоит из результатов тестов CPU, GPU, Памяти, UX. Каждой такой группе соответствует свой регулятор. По оси ординат откладывается количество условных единиц - баллов. Баллы помогают примерно оценить, какой регулятор смог продемонстрировать большую производительность на наборе тестов от AnTuTu. Точное количество баллов можно найти в Excel документе с таблицами.

Рассмотрим результаты, показанные на рисунке 6. Самые высокую производительность показал регулятор Performance, а самую низкую - Powersave, что ожидаемо. Стоит обратить внимание на результат одной из модификаций регулятора Schedutil - Alucardsched. Он сравнительно низкий по сравнению со Schedutil и другими модифицированными версиями, что является интересным фактом с учетом тестов энергопотребления, где этот регулятор показывал весьма низкую экономию энергии. Также стоит отметить, что остальные модификации остаются примерно на одном уровне по части производительности: их количество баллов практически не отличается. При этом Schedutil и его модификации не смогли обогнать такие алгоритмы, как Ondemand, Interactive и Userspace,

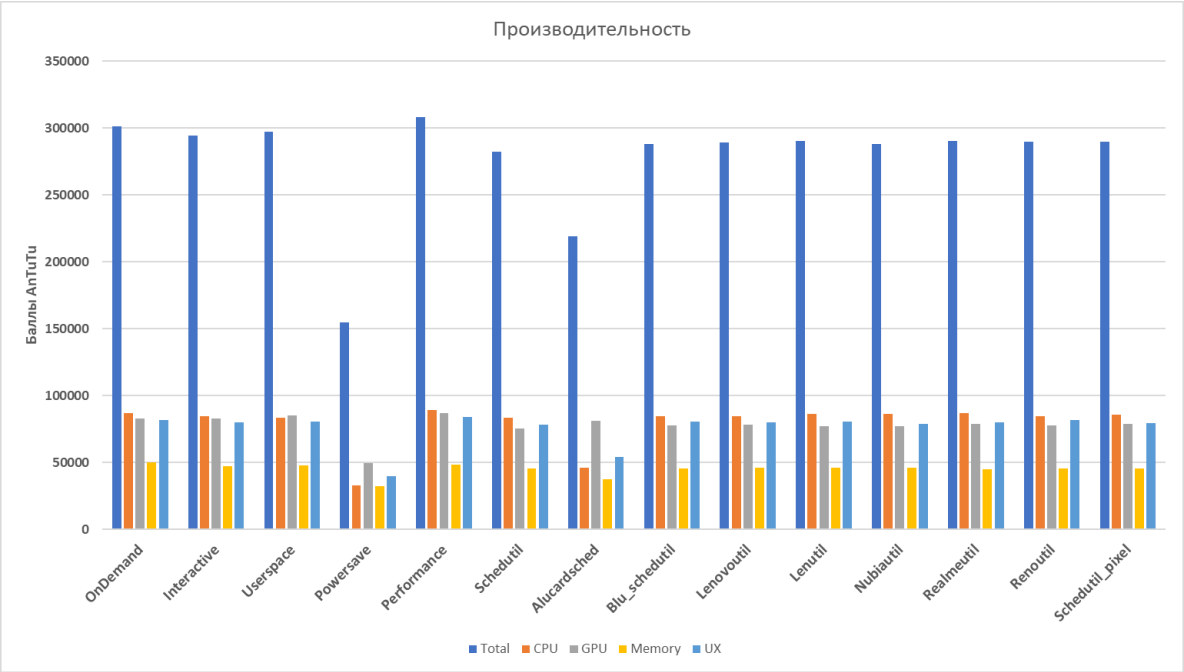

Рис. 6: Результаты тестов AnTuTu

которые находятся в ядре по умолчанию.

3.4.3. Общие выводы

По результатам проведенного анализа можно сделать следующие выводы:

1. Среди новых регуляторов были Schedutil и его модификации. Они показывали в основном меньшее потребление энергии, чем основные регуляторы вроде Ondemand и Interactive. При этом они практически никак не уступили в производительности при тестах AnTuTu. Исключением является регулятор Alucardsched.

2. Регулятор Alucardsched очень сильно выделяется на фоне других модификаций Schedutil, причем не в лучшую сторону. На всех тестах энергопотребления он демонстрировал одно из наибольших значений потребления энергии, в некоторых моментах даже доходя до уровня регулятора Performance (напомним, что он всегда выставляет максимальное значение частоты). Но при этом на тестах AnTuTu он показал один из наихудших результатов по производительности. Можно сделать вывод, что на всех проведенных тестах Alucardsched показал себя хуже всего.

3. Если говорить об остальных модификациях Schedutil, то можно сказать, что почти на всех тестах они показывают очень похожие

результаты. Явным исключением здесь будет тест по съемке видео, где отличия уже более заметны. Однако стоит отметить, что эти модификации потребляли больше энергии, чем сам регулятор Schedutil. При этом на тестах AnTuTu Schedutil показал не сильно меньшую производительность.

4. Заключение

В рамках данной работы были достигнуты следующие результаты:

1. Проведен обзор существующих подходов к оптимизации энергопотребления. Для этого:
   - изучено устройство подсистемы CPUFreq ядра Linux;
   - изучены регуляторы: Ondemand, Conservative, Performance, Powersave, Userspace, Interactive;
   - рассмотрены современные подходы: адаптивные алгоритмы, алгоритмы с машинным обучением, алгоритмы с предварительными вычислениями.
2. Проведена организация работы с тестовым стендом. Для этого:
   - в качестве устройства для тестирования был выбран смартфон Xiaomi Redmi Note 8 Pro;
   - подготовлена инфраструктура для загрузки новых алгоритмов DVFS: собрано и прошито ядро, установлена прошивка, поддерживающая установку сторонних ядер Android.
3. Проведена соответствующая подготовка к экспериментам:
   - определены метрики сравнения выбранных алгоритмов;
   - выбрана утилита для автоматизации тестирования - Monkeyrunner;
   - написаны сами тесты и выложены в открытый репозиторий GitHub[1];
   - определено, каким образом будет проводиться эксперимент;
   - определены способы оценки потребляемой энергии и примерной оценки производительности.
4. Проведены соответствующие эксперименты, результаты обработаны[2]. По полученным значениям сделаны определенные выводы об энергопотреблении и производительности тех или иных алгоритмов DVFS.

[1] GitHub репозиторий с тестами.
[2] Результаты вычислений.

Благодарность

Выражаю благодарность выпускнику Математико-Механического факультета Программной Инжерении Богданову Евгению Алексеевичу за содействие в подготовке тестового стенда.

5. Список литературы

[1] Number of smartphone users worldwide from 2016 to 2021:
https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] Mobile Operating System Market Share Worldwide Oct 2019 - Oct 2020:
https://gs.statcounter.com/os-market-share/mobile/worldwide

[3] CPUFreq Documentation:
https://www.kernel.org/doc/Documentation/cpu-freq/

[4] CPUFreq Governors:
https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[5] CPU power consumption reduction in android smartphone:
https://ieeexplore.ieee.org/abstract/document/7315080

[6] A powersaving DVFS algorithm based on operational intensity for embedded systems:
https://www.jstage.jst.go.jp/article/elex/12/3/12_12.20141128/_article/-char/ja/

[7] Usage History-Directed Power Management for Smartphones:
https://link.springer.com/chapter/10.1007/978-3-319-27119-4_20

[8] Reducing the energy consumption using dvfs performance optimizing scheme:
https://eprajournals.com/jpanel/upload/714pm_11.K.%20Poornambigai-798.pdf

[9] Learning-Directed Dynamic Voltage and Frequency Scaling Scheme with Adjustable Performance for Single-Core and Multi-Core Embedded and Mobile Systems:
https://www.mdpi.com/1424-8220/18/9/3068

[10] Energy-Efficient Control of Mobile Processors Based on Long Short-Term Memory:
https://ieeexplore.ieee.org/abstract/document/8737670/

[11] System-Level Power Management using Online Machine Learning for Prediction and Adaptation
https://eprints.soton.ac.uk/399995/

[12] Autonomous Power Management for Embedded Systems Using a Non-linear Power Predictor:
https://ieeexplore.ieee.org/abstract/document/8049763

[13] Hardware-software interaction for run-time power optimization: A case study of embedded Linux on multicore smartphones:
https://ieeexplore.ieee.org/abstract/document/7273508

[14] An Adaptive and Integrated Low-Power Framework for Multicore Mobile Computing:
https://www.hindawi.com/journals/misy/2017/9642958/

[15] MobiCore: An Adaptive Hybrid Approach for Power-Efficient CPU Managment on Android Devices:
https://ieeexplore.ieee.org/abstract/document/8226044/

[16] Application-Specific Performance-Aware Energy Optimization on Android Mobile Devices:
https://ieeexplore.ieee.org/abstract/document/7920823/

[17] Mobile Vendor Market Share Russian Federation:
https://gs.statcounter.com/vendor-market-share/mobile/russian-federation

[18] Adding Governors to your kernel:
https://thealaskalinuxuser.wordpress.com/2016/06/10/adding-governors-to-your-kernel/

[19] How to add a CPU Governor:
https://riptutorial.com/android/example/28283/how-to-add-a-cpu-governor

[20] Статьи, отобранные в результате системного обзора:
https://docs.google.com/document/d/1NvfuAtoKw2AmAP8gbgept4rf-yPuXz4M9UiWjSxmHZk/edit

[21] Extracted Data SEIM 2020:
https://docs.google.com/spreadsheets/d/1JDhrmRp6oMgL_sZcg5oePtcB_Qt_r5WKD9a7jUECm40/edit#gid=0

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

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

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

[25] Open GApps. Cite with the latest version of pre-built Open Google apps:
https://opengapps.org/

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

[27] 4pda forum. Magdisk open source utility discussion:
https://4pda.ru/forum/index.php?showtopic=774072

[28] Анализ проведения экспериментов:
https://1drv.ms/w/s!AuRU3kRUzNQhmxYSsqVUu4OPzr1A?e=Q5U7fG

[29] Adreno Profiler:
https://developer.qualcomm.com/software/adreno-gpu-profiler

[30] GameBench:
https://www.gamebench.net/

[31] XDA-Forum - governors:
https://forum.xda-developers.com/t/
ref-guide-most-up-to-date-guide-on-cpu-governors-i-o-schedulers-and-more.
3048957/

[32] Собранные алгоритмы DVFS:
https://1drv.ms/w/s!AuRU3kRUzNQhmxYSsqVUu4OPzr1A?e=Q5U7fG

[33] Schedutil information:
https://lkml.org/lkml/2016/3/17/420

[34] Android Debug Bridge:
https://developer.android.com/studio/command-line/adb

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

[36] UIAutomator:
https://developer.android.com/training/testing/ui-automator

[37] Robotium:
https://github.com/RobotiumTech/robotium

[38] Selendroid:
http://selendroid.io/

[39] Monkeyrunner:
https://developer.android.com/studio/test/monkeyrunner

[40] Flappy Bird:
https://en.wikipedia.org/wiki/Flappy_Bird

[41] International Conference on Event-based Control,Communication, and
Signal Processing:
https://ebccsp2021.org/

[42] Trial Xtreme 3:
https://4pda.ru/forum/index.php?showtopic=410812