Санкт-Петербургский государственный университет

БОГДАНОВ Егор Дмитриевич

Выпускная квалификационная работа

Энергоэффективное планирование задач в ОС Android

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2018 «Программная инженерия»*

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

Консультант:
ст. преп. каф. системного программирования С.Ю. Сартасов

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

Санкт-Петербург
2022

Saint Petersburg State University

Egor Bogdanov

Bachelor's Thesis

Energy efficient task scheduling in Android OS

Education level: bachelor

Speciality *09.03.04 «Software Engineering»*

Programme *CB.5080.2018 «Software Engineering»*

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

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

Saint Petersburg
2022

Оглавление

Введение

В наше время смартфоны, умные часы, ноутбуки и другие умные устройства, которые упрощают жизнь человеку, все сильнее проникают в нашу жизнь. Если эта тенденция сохранится, то в ближайшем будущем подавляющее большинство населения планеты будет иметь персональный компьютер в кармане. В настоящее время самая популярная используемая в таких устройствах операционная система — это Android.

Одна из главных проблем в использовании мобильных устройств — малое время работы устройств в связи с высоким расходом энергии аккумулятора, от которого работает смартфон. Чем меньше расход аккумулятора, тем дольше работоспособность устройства. Производители данных устройств пытаются адаптироваться и улучшать аппаратную и программную составляющую устройств.

Современные процессоры могут работать на разных уровнях частот и энергопотребление процессора на разных уровнях частот тоже разное. В ядре Linux, а следовательно и в Android, имеется подсистема CPUFreq. Она отвечает за регулировку частоты процессора, используя алгоритмы, именуемые алгоритмами DVFS (Dynamic Voltage and Frequency Scaling). Проблемой этих алгоритмов является то, что они не имеют информации об исполняемых на процессоре задачах.

Улучшить энергоэффективность процессора можно, если при планировании задач учитывать информацию о его энергопотреблении и особенностях архитектуры. Создание планировщика, который был бы эффективен в производительности и энергопотреблении наравне с существующими решениями, — довольно трудоёмкая задача, можно найти способы улучшения энергоэффективности использующихся планировщиков. Так как существуют особенности архитектуры, позволяющие это сделать, в этой работе такая модификация планировщика задач создаётся.

1. Постановка задачи

Цель работы — создать энергоэффективную модификацию актуального планировщика операционной системы Android и протестировать её на реальном устройстве.

Для достижения цели были поставлены следующие задачи:

- Найти актуальный планировщик задач и сделать обзор принципов его работы;
- Предложить и реализовать модификацию для повышения его энергоэффективности;
- Определить методологию тестирования и создать инфраструктуру для загрузки модифицированной версии планировщика задач;
- Провести тестирование и сравнить результаты обычного и модифицированного планировщика;

2. Обзор предметной области

В последние 10 лет активно велись исследования в области энергоэффективного планирования задач. Существуют планировщики, использующие как алгоритмы, основанные на разных эвристиках [23, 12], так и методы машинного обучения [13] и генетические алгоритмы [17]. Существенная проблема большинства этих планировщиков заключается в том, что они никак не интегрированы с операционной системой Android. Однако в 2015 году компания ARM представила планировщик задач под названием Energy Aware Scheduling (EAS) [11]. Он нацелен на распределение задач по процессорам с учётом как потенциального энергопотребления, так и требуемой производительности. EAS внедрён в ядро Linux начиная с версии 5.0 и вместе с этим уже используется на мобильных устройствах с версией ОС Android 8 и выше [11].

2.1. Energy Aware Scheduling

Планировщик по умолчанию в ядре Linux, Completely Fair Scheduler (CFS) [5], реализует политику размещения задач, ориентированную на производительность. EAS добавляет к этому планировщику политику размещения задач на основе энергосбережения, которая оптимизирует энергопотребление, управляя размещением задач и свободной мощностью ЦП [11, 16, 10]. EAS работает, когда система имеет низкую или среднюю общую загрузку, а исходная политика CFS работает при полной загрузке системы, поскольку EAS не дает преимущества в энергопотреблении, когда все ЦП перегружены. EAS предназначен для многокластерных геретогенных систем с неперекрывающимися кривыми мощности/производительности с разными типами ядер ЦП и состояниями простоя ЦП с отключением питания для каждого кластера и/или для каждого ЦП вроде архитектуры big.LITTLE [2].

EAS расширяет несколько различных подсистем, присутствующих в ядре. Большая часть EAS находится в файле `kernel/sched/fair.c` и представляет собой алгоритм, отвечающий за решения о размещении задач. Этот модуль строит необходимые структуры, содержащие энер-

гетические метрики, которые используются для расчета энергоэффективности. Код расширяет часто используемую политику планирования CFS и не затрагивает другие политики. Некоторые из существующих функций в CFS были модифицированы с учетом энергопотребления за счет учета возможных затрат энергии на планирование задач и управление процессорами, на которых выполняются задачи. Таким образом, EAS влияет на балансировку нагрузки и решения по упаковке задач. Ключевым моментом стало то, что смещать размещение задач в пользу энергосберегающей работы имеет смысл только тогда, когда имеется свободная мощность ЦП. При отсутствии свободных мощностей система обычно находится в состоянии, когда производительность является решающим фактором. Преднамеренный уклон в сторону энергоэффективной работы в таких случаях может привести к снижению производительности.

2.1.1. Энергетическая модель

Чтобы предоставить планировщику возможность оценивать требования к мощности и производительности задач, требуется введение новых структур данных, основанных на концепции модели энергопотребления. С этой целью был введен общий механизм для предоставления планировщику модели энергопотребления, данные которой состоят из:

- потребляемой мощности для каждого поддерживаемого P-состояния (Operating Performance Points (OPP) — пара значений частоты и соответствующего напряжения)
- энергопотребления для каждого C-состояния (состояние управления питанием в режиме ожидания)

Модель содержит данные только для процессоров и кластеров. Энергия обслуживания кластера, которая может варьироваться в зависимости от конкретной архитектуры, добавляется к энергии, связанной с каждым ЦП, для получения точных оценок. Данные модели энергопотребления в дереве устройств передаются планировщику через функции из файла `kernel/sched/energy.c`.

2.1.2. Отслеживание загрузки

Отслеживание нагрузки по задачам в CFS (и, соответственно, EAS) реализовано с использованием метода Per-Entity Load Tracking (PELT) [22]. Итоговая загрузка задачи зависит от её использования (utilization) (показатель, основанный на запланированном времени исполнения) и общей загрузки (показатель, основанный на времени, пока задача была в очереди на исполнение, и времени исполнения в предыдущие разы). Тот факт, что историческое поведение задачи со временем становится все менее значимым при попытке сделать выводы о её будущих требованиях к вычислительным ресурсам, учитывается с помощью использования геометрического ряда для каждого из двух показателей.

Для более точного размещения задачи также оцениваются с помощью нового алгоритма Window Assisted Load Tracking (WALT) [15]. В этом алгоритме имеется пять скользящих временных окон по 20 мс для подсчета исторических рабочих нагрузок задач. Рабочая нагрузка является фактическим временем исполнения задачи за временное окно, нормализированое с учётом мощности процессора на момент исполнения задачи. После подсчета пяти исторических рабочих нагрузок алгоритм WALT выбирает максимум между самым последним значением рабочей нагрузки и средним значением рабочих нагрузок за последние пять окон. Это значение используется в качестве прогнозируемой рабочей нагрузки задачи в следующем скользящем окне.

2.1.3. Перегруженные процессоры

Случаи использования, в которых EAS может помочь больше всего, связаны с низкой/средней загрузкой ЦП. Всякий раз, когда выполняются длительные задачи, требующих ресурсов ЦП, они требуют всей доступной мощности ЦП, и планировщик мало что может сделать для экономии энергии без серьезного ущерба производительности. Чтобы избежать снижения производительности с помощью EAS, процессоры помечаются как «перегруженные», как только они используются более чем на 80% своей вычислительной мощности. Пока в системе нет

перегруженных ЦП, балансировка нагрузки CFS отключена, и EAS переопределяет код размещения задач. EAS, скорее всего, загрузит наиболее энергоэффективные процессоры системы больше, чем другие, если это можно сделать без ущерба для производительности. Таким образом, балансировщик нагрузки отключен, чтобы он не нарушал энергоэффективное размещение задач, определённое EAS. Это безопасно делать, когда система не перегружена, поскольку загруженность менее 80% означает, что:

- на всех ЦП есть некоторое время простоя, поэтому оценки использования, используемые EAS, вероятно, точно отражают «размер» различных задач в системе;
- всем задачам предоставлены необходимые им мощности ЦП вне зависимости от их приоритета;
- поскольку есть резервная мощность, все задачи регулярно блокируются или переходят в спящий режим, поэтому нагрузка на всех ЦП будет сбалансирована при размещении задач.

Как только один ЦП превысит пороговую точку 80%, для всего кластера ЦП поднимается флаг перегрузки, EAS отключается, а балансировщик нагрузки CFS снова включается. Таким образом, планировщик возвращается к алгоритмам на основе нагрузки для пробуждения задач и балансировки нагрузки в условиях, связанных с процессором.

2.1.4. Размещение задач

EAS модифицирует функцию выбора процессора для исполнения задачи. Когда система не перегружена, задачи обычно размещаются следующим образом:

1. Задачи сначала проходят через `find_best_target`, чтобы определить для них «лучший» процессор. Любые процессоры, у которых свободной мощности не хватает для исполнения задачи, отбрасываются.

2. `find_best_target` попытается выбрать два процессора для задачи, основной и запасной, за исключением случаев, когда чувствительная к задержке задача находит доступный простаивающий ЦП. Задачи, которые чувствительны к задержке или имеют повышенное использование, начинают искать процессор в кластере процессоров с наибольшей мощностью. Все остальные задачи начинают поиск, начиная с группы, содержащей процессоры с наименьшей мощностью.

   (a) Задачи, чувствительные к задержке, будут выбирать первый найденный простаивающий ЦП. Если свободного процессора нет, они сначала выберут ЦП с наибольшим объемом свободных вычислительных мощностей, а в качестве альтернативы они также выберут ЦП с наименьшим использованием, когда эта задача будет помещена туда. Эти две опции выражают динамику распространения/упаковки для этого класса задач.

   (b) Задачи, не чувствительные к задержке, выберут основной целью активный ЦП, который имеет наименьшую максимальную мощность и наибольшую свободную мощность после добавления задачи. Цель этой стратегии - попытаться распределить задачи в наиболее энергоэффективном кластере ЦП. Запасным ЦП будет выбран простаивающий ЦП (если он есть), который имеет наименьшую максимальную мощность и самое неглубокое состояние бездействия.

3. Если задача чувствительна к задержке, она размещается на первом обнаруженном простаивающем ЦП.

4. Для всех других задач выбранные процессоров перед использованием оцениваются на предмет компромисса между энергопотреблением и производительностью. Запасной процессор оценивается только в том случае, если на основном процессоре оказывается невозможным разместить задачу.

2.2. Модификаций EAS

Для поиска модификаций EAS использовался сервис “Google Scholar” с поисковыми запросами по ключевым словам: energy aware scheduling, energy aware scheduler, EAS, android, linux, ARM, big.LITTLE. Из полученных результатов были отобраны работы, темы которых были прямые модификации конкретного планировщика EAS. На втором этапе были изучены другие работы авторов отобранных статей, а также работы, в которых отобранные на первом шаге статьи цитировались. Несмотря на то, что поисковые запросы выдавали большое количество статей, тексты которых связаны с энергоэффективным планированием задач, большинство из них не являлись модификациями EAS. Таким образом был найден список отобранных статей, посвященных улучшению работы планировщика.

Han at al. [14] представляют в своей работе Learning EAS. Авторы применили метод градиентного обучения с подкреплением к двум значениям, которые в оригинальном планировщике являются константными: TARGET_LOAD, значение, определяющее целевую нагрузку ядер, при превышении которого система становится перегруженной, и sched_migration_cost, значение, влияющее на оценку энергопотребления при перемещении задачи с одного процессора на другой. Learning EAS улучшил энергопотребление на 2,3–5,7%, результаты hackbench (тесты производительности планирования процессов) на 2,8–25,5%, время входа приложений на 4,4–6,1% и время входа приложений при высокой нагрузке на ЦП на 9,6-12,5% соответственно по сравнению с EAS. Авторы предполагают, что можно достичь улучшения энергопотребления и производительности путём применения машинного обучения к различным подсистемам ядра Linux.

Zhang at al. [24] улучшают точность прогнозирования рабочей нагрузки задач, энергоэффективность выбора ядра центрального процессора и балансировку нагрузки в своей версии Wearable-Application-Optimized Energy Aware Scheduler (WAEAS). Авторы заменяют обычное среднее значение рабочих нагрузок в алгоритме WALT на скользящую

среднюю с целью увеличение влияния краткосрочных нагрузок в прогнозируемой загрузке задачи. Вместе с этим исследователи изменили приоритеты выбора ядра: для задач, чувствительных к задержке, выбирается не первое найденное ядро в состоянии простоя, а ищется ядро с меньшей максимальной мощностью и менее глубоким состоянием простоя. Для остальных задач используется стратегия компоновки задач, при которой планировщик не равномерно распределяет нагрузку по ядрам LITTLE, а пытается разместить задачи на наименьшем количестве ядер. Основная идея этой стратегии — освободить некоторые ядра от нагрузки для того, чтобы перевести их в состояние простоя и потенциально понизить энергопотребление. Авторы также меняют алгоритм перебалансировки задач CFS: в новом варианте задачи перераспределяются только внутри кластера. Исследователи утверждают, что этот метод снижает количество бессмысленных миграций задач между кластерами, тем самым уменьшая энергопотребление системы на основе гарантированной производительности системы. По результатам тестирования модифицированная версия EAS улучшила производительность на 5% и энергопотребление на 8% по сравнению с обычной версией. Единственным замечанием является то, что тесты проходили со всеми тремя модификациями, поэтому оценить влияние каждой отдельно взятой оптимизации не представляется возможным.

3. Улучшение алгоритма выбора процессора

3.1. Описание модификации

В ходе работ предшественников под руководством Сартасова Станислава Юрьевича и Граничина Олега Николаевича было выявлено, что энергопотребление кластера ядер LITTLE нелинейно зависит от количества работающих ядер на одинаковой частоте [25].

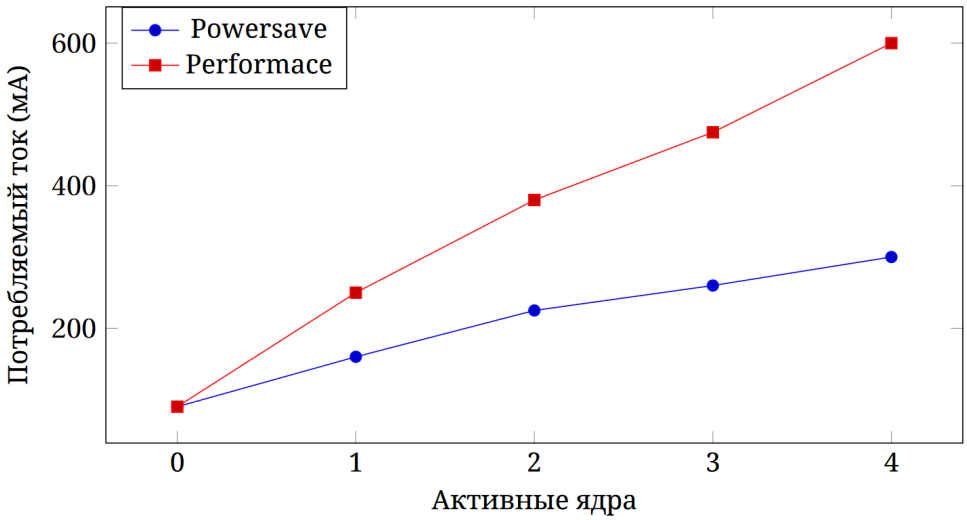

Рис. 1: Энергопотребление кластера ядер LITTLE в зависимости от числа активных ядер [25].

Аналогичное явление было обнаружено другими энтузиастами Android при исследовании системы на кристалле Exynos 5433 [7] (Рис. 2).

Основываясь на этих наблюдениях, модификация EAS с более точным учитыванием трат энергии при выборе между выводом имеющегося простаивающего процессора кластера LITTLE из текущего состояния простоя и переводом кластера ядер LITTLE на более высокий уровень мощности при распределении задач может увеличить энергоэффективность без значительных потерь производительности.

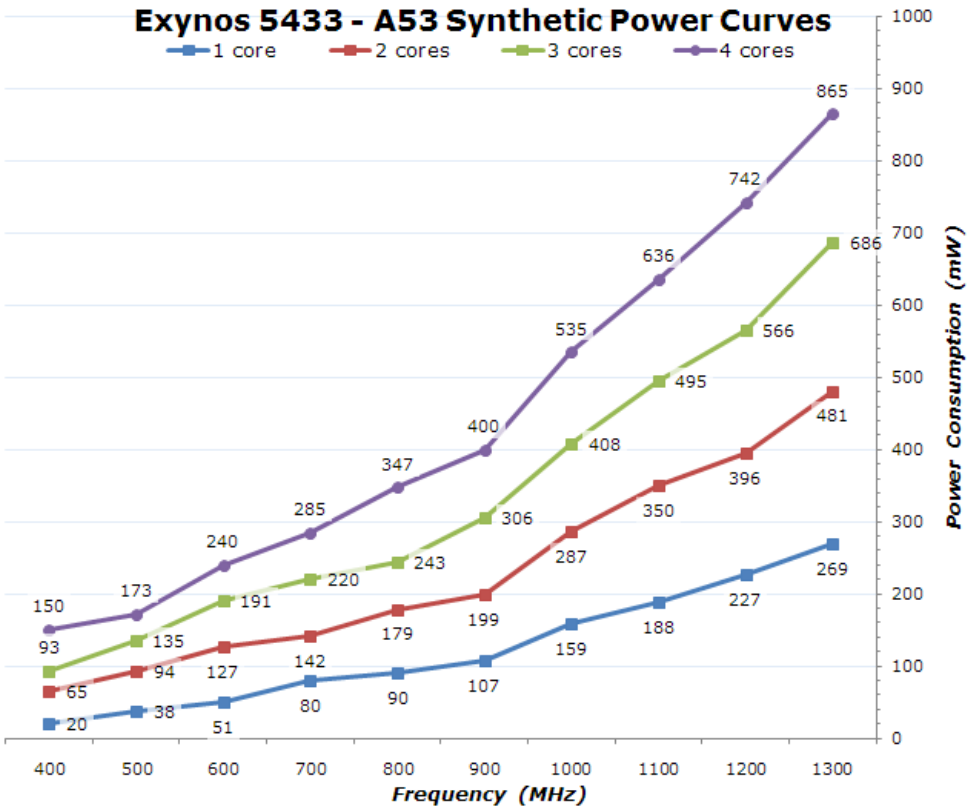

Рис. 2: Графики зависимости энергопотребления ядер от их частоты и количества [7].

3.2. Реализация модификации

Модификация затрагивает алгоритм выбора процессора только для нечувствительных к задержке задач, так как в случае чувствительных к задержке задач добавление кода для исполнения может негативно влиять на их быстродействие. Новый алгоритм принимает `target_cpu` и `best_idle_cpu`, которые EAS выбрал для текущей задачи. Если они оба из кластера малых ядер и при размещении задачи на `target_cpu` повысится частота кластера, то в этом случае рассчитывается новая мощность ядер кластера и на основе этой мощности рассчитывается энергопотребление системы. Затем рассчитывается энергопотребление системы в случае размещения задачи на `best_idle_cpu`. Результаты сравниваются и возвращается наиболее энергоэффективный процессор.

Ниже представлен псевдокод алгоритма:

Algorithm 1 Модификация EAS

Input: $target\_cpu, best\_idle\_cpu$
Output: $efficient\_cpu$

```
if target_cpu, best_idle_cpu ∈ LITTLE_CORE_CLUSTER then
    new_util ← cpu_util(target_cpu) + task_util
    if new_util < current_capacity(target_cpu) then
        old_cap ← current_capacity(target_cpu)
        new_cap ← calc_new_capacity(new_util)
        rising_cost ← calc_energy_consumption(target_cpu, new_cap)
        wake_cost ← calc_energy_consumption(best_idle_cpu, old_cap)
        if rising_cost < wake_cost then
            return target_cpu
        end if
        return best_idle_cpu
    end if
end if
```

4. Инфраструктура для загрузки модифицированных версий ядер

4.1. Xiaomi Redmi 9A

В качестве тестового стенда использовался смартфон Xiaomi Redmi 9A со следующими ключевыми характеристиками:

- ОС Android 10 с оболочкой MIUI 12;
- Версия ядра Linux 4.9.190;
- Восьмиядерный процессор MediaTek MT6762G Helio G25 (4x2.0 GHz Cortex-A53 & 4x1.5 GHz Cortex-A53);
- 32 гигабайта внутренней памяти;
- 2 гигабайта оперативной памяти;

Устройства Xiaomi отличаются тем, что для работы с ними разработано много официальных инструментов, а также существует большое сообщество людей, занимающихся их модификациями. К тому же на выбранном устройстве установлена одна из последних версий операционной системы Android, а также имеет приемлемую цену и современный многоядерный процессор [19].

4.2. Особенности компиляции и установки ядер

Для установки сторонних ядер на устройство необходим разблокированный загрузчик и стороннее рекавери, способное устанавливать сторонние ядра. Для разблокировки загрузчика была использована официальная утилита Mi Unlock Tool [18]. В качестве стороннего рекавери было выбрано OrangeFox Recovery [21], так как оно поддерживает выбранное устройство и обладает необходимой функциональностью.

Xiaomi предоставляет официальный репозиторий [20], в котом содержатся исходные коды ядер выпущенных устройств, в том числе и

Xiaomi Redmi 9A. Существенной проблемой стало, что инструкция по сборке ядер, находившаяся в этом репозитории, оказалась недостоверной и процесс компиляции прекращался с ошибкой. Благодаря помощи энтузиаста из сообщества разработчиков Android были выявлены следующие особенности:

- Для корректной сборки ядра необходимо использовать инструменты компиляции AOSP GCC 4.9 и AOSP Clang 6.0;
- В коде ядра отсутствовали необходимые для компиляции файлы `fw_helitai_v0e.i` и `fw_sample.i`. Создание пустых файлов с такими именами решило проблему компиляции и в дальнейшем не повлияло на работоспособность ядра и операционной системы в целом;
- Для загрузки ядра на устройство необходимо с помощью утилиты AnyKernel3 [4] из скомпилированных файлов `Image.gz`, `dtb.img` и `dtbo.img` создать архив, из которого утилита восстановления сможет извлечь и установить стороннее ядро.

5. Эксперименты

Важной частью работы является проверка энергоэффективности модифицированной версии планировщика. Авторы других модификаций оценивали планировщик с помощью тестов, имитирующих реальное использование устройства. Было принято решение взять за основу тестовые сценарии, реализованные студентом СПбГУ Александром Божнюком в рамках его учебной практики[1]. Эти тесты были адаптированы под целевое устройство, а также в них был добавлен расчёт потраченной процессором энергии.

5.1. Инструменты и сценарии тестирования

Monkeyrunner

Для автоматизации тестирования используется утилита Monkeyrunner [9]. Эта утилита разработана компанией Google и поставляется вместе с инструментами для разработчиков Android SDK. Monkeyrunner предоставляет API для написания программ на языке Python. Эти программы позволяют управлять устройством Android при помощи компьютера, используя технологию Android Debug Bridge [8].

Благодаря этой утилите можно имитировать взаимодействие пользователя с устройством и точно повторять последовательность действий при проведении тестов, обеспечивая их идентичность между собой. Также Monkeyrunner не требует доступа к исходному коду тестируемого приложения.

Тестовые сценарии

Для проведения тестирования был выделен ряд тестовых случаев — возможных сценариев использования смартфона. Часть из них имитирует сложные вычислительные задачи, например, игры или съёмка видео, которые выражаются в высокой нагрузке на процессор, а другая

[1] https://github.com/bozhnyukAlex/dvfs_testing

часть имитирует более простые задачи, такие как просмотр видео или набор текстовых сообщений.

Ниже приведено описание тестовых сценариев:

- **Camera test**: запуск камеры смартфона и съёмка видео в течение 3 минут;
- **Typing test**: запуск приложения для заметок Notes, создание новой заметки и набор текста в течение 3 минут;
- **Flappy Bird test**: запуск игрового приложения Flappy Bird и имитация игровой деятельности при помощи касаний экрана смартфона в течение 3 минут;
- **Trial Xtreme test**: запуск игрового приложения Trial Xtreme 3 и имитация игровой деятельности при помощи касаний экрана смартфона в течение 3 минут;
- **Video test**: запуск приложения VCL player for Android и воспроизведение видео в течение 3 минут;

5.2. Методология тестирования

Для повышения точности и объективности экспериментов был продуман ряд мер, которые минимизируют влияние внешних факторов на результаты:

- Отключены внешние модули (Wi-Fi, 3G, GPS, Bluetooth) и включён авиарежим;
- В настройках операционной системы отключена работа фоновых приложений;
- Были удалены все сторонние приложения, не относящиеся к проводимым тестам;
- Каждый тест исполнялся 13 раз. Сперва устройство ”прогревалось” на первых трёх запусках. Результаты следующих 10 тестов

усреднялись, чтобы уменьшить влияние возможных сторонних активностей.

- Между запусками наборов тестов выдерживалась пауза в 5 минут, чтобы перед запуском нового набора тестов корректно завершились все связанные с предыдущим запуском фоновые процессы, а само устройство остыло до исходной температуры.

5.3. Оценка потребляемой энергии

Подсистема ядра Linux CPUFreq [6] предоставляет различную статистику, связанную с частотой кластеров ядер. Необходимой информацией из этой является информация о времени, проведённом кластером на каждой из поддерживаемых частот. В начале и в конце каждого теста производилось считывание этой статистики и высчитывалась разница, которая использовалась позже при подсчёте затраченной энергии.

Также из смартфона был извлечён файл `power_profile.xml`, в котором производители устройства предоставляют информацию о энергопотреблении модулей телефона. Данный файл содержит константы энергопотребления кластеров процессора при их работе на поддерживаемых частотах в течение 10 миллисекундного периода. Найденные константы представлены в таблице 1.

Таким образом оценка электропотребления в миллиампер-часах высчитывалась по следующей формуле:

$$power\_consumption = \frac{\sum_{freq} time\_on\_freq(freq) \cdot consumption(freq)}{3600 \cdot 100}$$

где *time_ on_ freq(freq)* — время, проведённое на частоте *freq* из статистики CPUFreq, *consumption(freq)* — энергопотребление кластера на частоте *freq*.

A53 1.5GHz		A53 2.0GHz	
Frequency (Hz)	Current (ma)	Frequency (Hz)	Current (ma)
1500000	94.42	2001000	409.51
1429000	84.73	1961000	396.05
1367000	73.16	1927000	384.1
1314000	60.33	1897000	365
1261000	56.65	1868000	340.66
1208000	84.73	1838000	249.74
1155000	45.77	1809000	230.72
1102000	43.55	1779000	210.44
1050000	39.42	1750000	157.17
948000	37.46	1617000	116.49
846000	35.99	1484000	90.9
745000	34.29	1351000	72.9
643000	33.34	1218000	66.1
542000	30.66	1085000	58.1
501000	30.07	979000	53.36
400000	27.94	900000	50.38

Таблица 1: Константы энергопотребления Xiaomi Redmi 9A

Оценка производительности

Наряду с оценкой энергоэффективности необходимо проверить, что предложенный алгоритм не ухудшил производительность устройства. Для этой оценки было принято решение использовать приложение AnTuTu Benchmark [3]. Это приложение является самым популярным среди приложений, тестирующих производительность [1], и после после ряда замеров выставляет баллы устройству по 5 различным категориям: CPU, GPU, RAM, IO, UX. В таблице 2 представлены результаты работы утилиты для обычной и модифицированной версии EAS:

	CPU	GPU	RAM	IO	UX	Total
Обычный EAS	15008	12148	6599	3126	8396	45277
Модифицированный EAS	14927	12021	6715	3012	8531	45206

Таблица 2: Баллы AnTuTu Benchmark

5.4. Анализ результатов

Результаты энергопотребления проведённых тестов представлены в таблице 3. В первой колонке указаны результаты обычной версии EAS, во второй — результаты модифицированной версии EAS. В скобках указана разница энергопотребления в процентах.

	Обычный EAS	Модифицированный EAS
Camera test	21.598	21.007 (-2.7%)
Typing test	5.244	5.177 (-1.2%)
Flappy Bird test	5.459	5.293 (-3%)
Trial Xtreme test	15.667	15.23 (-2.8%)
Video test	4.959	4.949 ($\sim$0%)

Таблица 3: Энергопотребление планировщиков задач, значения указаны в миллиампер-часах

Из таблицы 3 видно, что в тесте с видео энергопотребление почти никак не изменилось. Это объясняется тем, что процесс воспроизведения мультимедиа является чувствительным к задержке, а выбор процессора для такого рода задач модификация не затрагивает из-за соображений производительности. В остальных случаях видно, что уменьшение энергопотребления достигает 3% при сохранении быстродействия. Таким образом предложенное улучшение алгоритма выбора процессора EAS успешно применено.

Заключение

В рамках данной дипломной работы были достигнуты следующие результаты:

- Проведён обзор актуального планировщика задач ОС Android:
  - найден и изучен планировщик задач EAS, использующийся в ОС Android с версии 8;
  - изучены модификации Learning EAS и WAEAS;
- На базе смартфона Xiaomi Redmi 9A налажен процесс компиляции и загрузки модифицированных ядер;
- Предложено и реализовано[2] улучшение алгоритма выбора энергоэффективного процессора:
  - в алгоритме учтена оценка энергопотребления при выборе между активным и простаивающим ядрами;
- Проведено тестирование модифицированного планировщика задач по 5 тестовым сценариям[3]. Предложенное улучшение не влияет на производительность и снижает энергопотребление на 1,4-3%.

[2]https://github.com/thofyb/eas_improvement
[3]https://github.com/thofyb/eas_testing

Список литературы

[1] 8 Android Benchmark Apps to Test Performance of Phone. — URL: https://geekflare.com/top-android-performance-check-apps/ (online; accessed: 06.05.2022).

[2] ARM big.LITTLE. — URL: https://www.arm.com/technologies/big-little (online; accessed: 06.05.2022).

[3] AnTuTu Benchmark official cite. — URL: https://www.antutu.com/en/index.htm (online; accessed: 06.05.2022).

[4] AnyKernel3 - Flashable Zip Template for Kernel Releases with Ramdisk Modifications. — URL: https://github.com/osm0sis/AnyKernel3 (online; accessed: 06.05.2022).

[5] CFS Scheduler. — URL: https://www.kernel.org/doc/html/latest/scheduler/sched-design-CFS.html (online; accessed: 06.05.2022).

[6] CPUFreq Documentation. — URL: https://www.kernel.org/doc/Documentation/cpu-freq (online; accessed: 06.05.2022).

[7] Cortex A53 - Performance and Power - ARM A53/A57/T760 investigated - Samsung Galaxy Note 4 Exynos Review. — URL: https://www.anandtech.com/show/8718/the-samsung-galaxy-note-4-exynos-review/4 (online; accessed: 06.05.2022).

[8] Developers Android. Android Debug Bridge (adb) overview page. — URL: https://developer.android.com/studio/command-line/adb (online; accessed: 06.05.2022).

[9] Developers Android. Monkeyrunner tool overview page. — URL: https://developer.android.com/studio/test/monkeyrunner (online; accessed: 06.05.2022).

[10] EAS Overview and Integration Guide. — URL: https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Open%20Source/energy-aware-scheduling/eas_overview_and_integration_guide_r1p6.pdf (online; accessed: 06.05.2022).

[11] Energy Aware Scheduling. — URL: https://developer.arm.com/tools-and-software/open-source-software/linux-kernel/energy-aware-scheduling (online; accessed: 06.05.2022).

[12] Energy-aware scheduling for improving manufacturing process sustainability: A mathematical model for flexible flow shops / Alessandro AG Bruzzone, Davide Anghinolfi, Massimo Paolucci, Flavio Tonelli // CIRP annals. — 2012. — Vol. 61, no. 1. — P. 459–462.

[13] Esmaili Amirhossein, Pedram Massoud. Energy-aware scheduling of jobs in heterogeneous cluster systems using deep reinforcement learning // 2020 21st International Symposium on Quality Electronic Design (ISQED) / IEEE. — 2020. — P. 426–431.

[14] Han Junyeong, Lee Sungyoung. Performance improvement of linux CPU scheduler using policy gradient reinforcement learning for android smartphones // IEEE Access. — 2020. — Vol. 8. — P. 11031–11045.

[15] Introduce Window Assisted Load Tracking. — URL: https://lwn.net/Articles/704903/ (online; accessed: 06.05.2022).

[16] Kernel.org: Energy Aware Scheduling. — URL: https://www.kernel.org/doc/html/latest/scheduler/sched-energy.html (online; accessed: 06.05.2022).

[17] Kumar Neetesh, Vidyarthi Deo Prakash. A GA based energy aware scheduler for DVFS enabled multicore systems // Computing. — 2017. — Vol. 99, no. 10. — P. 955–977.

[18] MIUI official cite. Unlocking bootloader instruction. — URL: http://www.miui.com/unlock/index_en.html (online; accessed: 06.05.2022).

[19] Mediatek official cite. MediaTek Helio G25 Series page. — URL: https://www.mediatek.com/products/smartphones-2/mediatek-helio-g25 (online; accessed: 06.05.2022).

[20] Official Xiaomi kernel sources. — URL: https://github.com/MiCode/Xiaomi_Kernel_OpenSource/ (online; accessed: 06.05.2022).

[21] OrangeFox recovery official cite. — URL: https://orangefox.download/ (online; accessed: 06.05.2022).

[22] Per-entity load tracking. — URL: https://lwn.net/Articles/531853/ (online; accessed: 06.05.2022).

[23] Safari Monire, Khorsand Reihaneh. Energy-aware scheduling algorithm for time-constrained workflow tasks in DVFS-enabled cloud environment // Simulation Modelling Practice and Theory. — 2018. — Vol. 87. — P. 311–326.

[24] Waeas: An optimization scheme of eas scheduler for wearable applications / Zhan Zhang, Xiang Cong, Wei Feng et al. // Tsinghua Science and Technology. — 2020. — Vol. 26, no. 1. — P. 72–84.

[25] Проблемы и перспективы использования стохастической аппроксимации для регулирования частоты процессора в Android OS / Станислав Юрьевич Сартасов, Евгений Алексеевич Богданов, Александр Сергеевич Божнюк et al. // Компьютерные инструменты в образовании. — 2021. — no. 2. — P. 26–40.