САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-Механический факультет
Кафедра Системного Программирования

Пелогейко Макар Андреевич

Создание регулятора состояния простоя для системы EAS

Отчет производственной практике

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

Санкт-Петербург
2022

Содержание

Введение

В наше время смартфоны с каждым годом глубже внедряются в жизнь людей, и их количество увеличивается. Вместе с этим повышается уровень технических характеристик, и постоянно решается вопрос времени автономной работы данных устройств.

В 5 версии ядра Linux появился планировщик EAS (Energy Aware Scheduler), который может использовать архитектуру гетерогенных ядер процессора для увеличения энергоэффективности. Во многих смартфонах стоят процессоры ARM с архитектурой big.LITTLE, то есть есть возможность использовать планировщик EAS. Учитывая, что ОС Android — одна из трех самых популярных операционных систем для смартфонов и она основана на ядре Linux, в данной работе рассматривается именно ОС Android.

В ОС Android есть несколько технологий для снижения энергопотребления, одной из которых является динамическое масштабирование напряжения и частоты (DVFS). DVFS представляет собой технологию, цель которой состоит в том, чтобы установить рабочее напряжение и тактовую частоту в соответствии с фактическим энергопотреблением процессора в данный момент, чтобы избежать избыточной производительности, которая может снизить энергопотребление.

В то же время смартфон не используется постоянно — есть длительные отрезки времени, когда он находится в состоянии ожидания или использует не все ресурсы. В это время для большего сохранения энергии возможно ввести весь ЦП (Central Processing Unit, CPU) или его отдельные ядра в состояние простоя (idle state). Эту задачу снижения энергопотребления решает система CPUIdle — система управления простоем ЦП [1].Она состоит из трех частей:

- регуляторы (governors) выбирают, в какое состояние простоя перевести ЦП;
- драйверы (drivers) передаёт решение регуляторов в аппаратную часть смартфона;
- ядро (core) общая платформа, связывающая систему воедино;

В ОС Android есть несколько вариантов стандартных регуляторов. Учитыввая, что EAS использует DVFS регулятор schedutil, разработанный специально для EAS, стоит посмотреть на возможность создания регулятора состояния простоя специально для системы EAS. Таким образом, в данной работе мы разрабатываем регулятор состояния простоя для планировщика EAS.

1 Цели и задачи

Цель: Создать регулятор состояния простоя для системы EAS и провести его апробацию
Задачи:
1 Сделать обзор подсистемы CPU Idle.
2 Сделать обзор работы EAS.
3 Сделать обзор работы schedutil.
4 Разработать регулятор.
5 Разработать методологию тестирования и подготовить окружение.
6 Провести тестирование.

2 Обзор устройства регуляторов простоя

Для рассмотрения регуляторов простоя сначала определим, что такое состояния простоя. Современные процессоры обычно могут переходить в состояния простоя, в которых выполнение программы приостанавливается. Поскольку часть аппаратного обеспечения процессора не используется в состояниях простоя, использование этих состояний позволяет снизить мощность, потребляемую процессором, и дает возможность сэкономить энергию [1]. Эти состояния различаются по времени входа в него, времени выхода и потребляемой энергии CPU в нем.

Перечислим некоторые часто используемые состояния простоя на примере устройства Samsung Galaxy Nexus 3 с Android 4.1.1 Ice Cream Sandwich. Смартфон оснащен процессором ARM dual core 1.2 GHz Cortex-A9 [6] (состояния перечисляются в порядке увеличения глубины простоя):

- C1 (WFI) — большинство таймеров процессора отключены. Задержка выхода из этого состояния составляет 4 мкс.
- C2 ((CPUs OFF, MPU + CORE INA) — ЦП выключен, блок защиты памяти (MPU) включен для защиты критически важных данных, а ядро неактивно. Задержка выхода из этого состояния составляет 1100 мкс.
- C3 (CPUs OFF, MPU + CORE Closed Switched with Retention) — аналогично состоянию C2, но ядро находится в режиме CSWR. Задержка выхода для этого состояния это 1200 мкс.
- C4 (CPUs OFF, MPU CSWR + CORE Open Switched Retention) — аналогично состоянию C3, но ядро находится в режиме OSWR. Задержка выхода из этого состояния составляет 1500 мкс.

Далее опишем Android OS API для работы регуляторов состояния простоя [2]:

- struct cpuidle_governor
  {
  char name[CPUIDLE_NAME_LEN];
  struct list_head governor_list;
  unsigned int rating;
  int (*enable) (struct cpuidle_driver *drv, struct cpuidle_device *dev);
  void (*disable) (struct cpuidle_driver *drv, struct cpuidle_device *dev);
  int (*select) (struct cpuidle_driver *drv, struct cpuidle_device *dev, bool *stop_tick);
  void (*reflect) (struct cpuidle_device *dev, int index);
  }; — структура регулятора простоя.

- CPUidle регистрирует регулятор функцией cpuidle_register_governor(), в качестве используемого может быть только 1 регулятор

  – int (*enable) (struct cpuidle_driver *drv, struct cpuidle_device *dev) — подготовка регулятора для обработки переданного ядра dev, в drv помимо драйвера должен лежать список состояний, в которые можно перевести данное ядро.

  – void (*disable) (struct cpuidle_driver *drv, struct cpuidle_device *dev) — остановит обработку данного ядра (dev) и освободит всю память, которая выделена под него.

  – int (*select) (struct cpuidle_driver *drv, struct cpuidle_device *dev, bool *stop_tick)

    * в drv находится драйвер и массив состояний;

    * в dev находится ядро, относительно которого мы сейчас принимаем выбор;

    * в bool *stop_tick — находится функция, определяющая, стоит ли остановить планировщик перед переходом CPU в выбранное состояние;

  Возвращаемое значение — индекс состояния из массива переданного в drv, в которое надо перевести данный CPU, или отрицательное число — код ошибки.
  Указатель на select в struct cpuidle_governor обязательно не равен NULL. Регулятор состояния простоя должен учитывать

задержку пробуждения процессора (power management quality of service, PM QoS) при выборе состояния простоя. Для получения текущей PM QoS задержки пробуждения регулятор простоя передает номер ядра в функцию
cpuidle_governor_latency_req(). После чего exit_latency состояния, индекс которого возвращает select, не может быть больше чем возвращаемое значение
cpuidle_governor_latency_req.

- void (*reflect) (struct cpuidle_device *dev, int index); оценивает точность выбора состояния, полученного вызовом select, на предмет использования этого состояния в дальнейшем.
- Поле name — строка, соответствующая названию регулятора простоя.
- Поле rating — рейтинг регулятора целое число. В качестве стандартного используется тот регулятор, у которого наибольший рейтинг[5].
- Поле governor_list — список всех регуляторов, который заполняется при передаче в функцию cpuidle_register_governor, описанной в файле cpuidle.h или governor.c

- Массив состояний должен быть отсортирован по "глубине" состояний - у состояния с индексом 0 минимальное значение
target_residency - самое неглубокое состояние. Данное поле содержится в
struct cpuidle_state, которая находится в
struct cpuidle_driver target_residency.
Поля в struct cpuidle_state, которые используются существующими регуляторами для вычислений, связанных с выбором состояния простоя:
  - target_residency - минимальное время с учетом времени входа в состояние, которое необходимо, чтобы сэкономить больше энергии нежели в менее глубоком состоянии
  - exit_latency - максимальное необходимое время для выхода ЦП из данного состояния и выполнения первой инструкции процессором.
- void (*enter) (struct cpuidle_device *dev, struct cpuidle_driver *drv, int index); указатель на функцию, вводящую CPU в выбранное состояние простоя. Находится в struct cpuidle_state и не может быть равен NULL.

3 Обзор системы EAS

Energy Aware Scheduling (или EAS) дает планировщику возможность прогнозировать влияние своих решений на энергию, потребляемую процессорами (или их ядрами). EAS полагается на модель энергопотребления (Energy Model, EM) ядер, чтобы выбрать энергоэффективное ядро ЦП для каждой задачи с минимальным влиянием на производительность.

EAS работает только с гетерогенными архитектурами ЦП (такими как Arm big.LITTLE), потому что именно здесь потенциал экономии энергии за счет планирования наиболее высок.

Определимся с некоторыми понятиями:

- энергия, измеряется в Джоулях — это ресурс, такой как батарея на устройствах с питанием
- мощность, измеряется в Ваттах (Джоуль / секунда) — это отношение энергии ко времени

Цель EAS - минимизировать потребление энергии и в то же время выполнить программную задачу. То есть мы хотим максимизировать:

$$\frac{\text{производительность[инструкция/сек]}}{\text{мощность[Ватт]}}$$

или минимизировать:

$$\frac{\text{энергия[Дж]}}{\text{инструкция}}$$

при этом сохраняя хорошую производительность. По сути, это цель оптимизации, альтернативная текущей цели планировщика, направленной только на производительность. Эта альтернатива рассматривает две цели: энергоэффективность и производительность.

Идея внедрения EM состоит в том, чтобы позволить планировщику оценивать последствия своих решений, а не слепо применять методы энергосбережения, которые могут иметь положительный эффект только на некоторых платформах. В то же время EM должна быть как можно более простым, чтобы минимизировать влияние задержки планировщика.

Таким образом, EAS меняет способ назначения задач ядрам. Когда планировщику пора решить, где должна выполняться задача, EM используется, чтобы выбрать из нескольких подходящих ядер для выполнения задачи то, которое, по прогнозам обеспечит наилучшее энергопотребление с минимальным ущербом производительности. Прогнозы, сделанные EAS, основаны на конкретных знаниях о топологии платформы, которые включают «емкость» ядер по производительности и соответствующие им затраты на электроэнергию.

4 Обзор регулятора schedutil

Schedutil - это регулятор cpufreq, который позволяет управлять выбором тактовой частоты процессора непосредственно из планировщика. Schedutil работает как промежуточный слой между планировщиком и платформой CPUFreq. Это позволяет планировщику реализовать регулятор политики CPUFreq самостоятельно, в основном заменяя устаревшие регуляторы, такие как ondemand, interactive и т. д. Тесная связь между планировщиком и выбором тактовой частоты обеспечивает лучшие общесистемные политики и улучшает как производительность, так и энергосбережение.

Schedutil можно активировать через обычные интерфейсы CPUFreq sysfs.

Schedutil предоставляет пользовательскому пространству повышающий и понижающий (Up/ Down) пороги регулирования. Пороги регулирования (превышающие предел физической задержки перехода) также необходимы для предотвращения частой (и потенциально опасной) смены тактовой частоты. Таким образом, schedutil решает задачу быстрого реагирования на внезапное увеличение использования и имеет небольшой гистерезис для кратковременных падений. Фактические значения (в микросекундах) можно настроить с помощью двух файлов sysfs регулятора: up_rate_limit_usec и down_rate_limit_usec

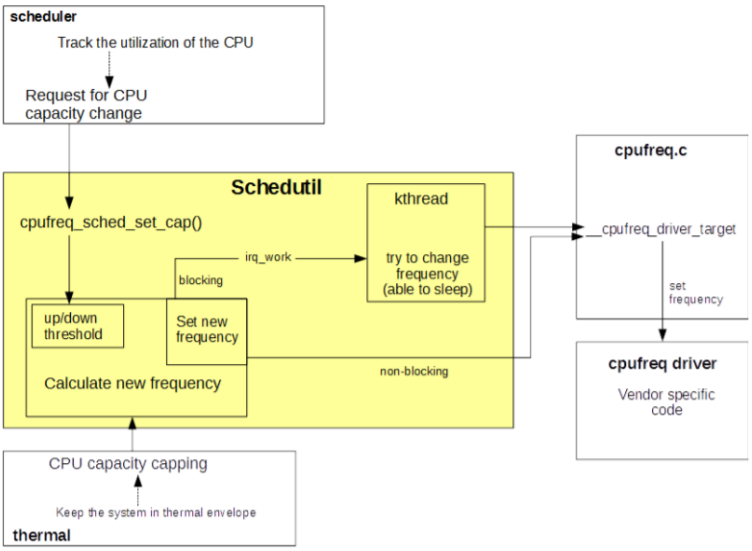

Рис. 1: Структура Schedutil.

5 Ход работы

5.1 Подготовка тестового стенда

В качестве тестового стенда был взят смартфон Samsung Galaxy s7 SM-G930FD. Сначала была поставлена прошивка Android 8.0 - SM-G930FXXU2ERD5 BTU - прошивка от производителя. Потом через Odin (программа для прошивки смартфонов) установлена lineage-17.1-20210409 (android 10), после чего взято open source ядро на основе elite_kernel с реализованной системой EAS[1] и для создания запускаемого zip используется проект lazyflasher ветки no-verity-opt-encrypt[2].

Было создано 2 Image-файла - скомпилированные проекты Android ядер со изначальный проект и модифицированный. Для сбора данных о энергопотреблении написаны скрипты get_freq_stats и get_battery_stats[3], которые должны находиться в памяти устройства в папке Downloads/IDLE_TEST.

5.2 Создание регулятора простоя

После анализа имеющегося кода было решено модифицировать имеющийся регулятор простоя menu, чтобы использовать возможности гетерогенной архитектуры. Поскольку у каждого ядра данного тестового стенда имеется всего 2 состояния простоя разрабатываемый алгоритм сводится к выбору между глубоким и не глубоким сном. Для наглядности и универсальности алгоритма была создана функция определяющая производительность ядра (get_core_class), номер которого регулятор получает в качестве одного из входных параметров. Данная функция делит ядра на 3 класса по производительности / энергопотреблению:

0. Little
1. Middle
2. Big

Так как в изначальном проекте не удалось найти Energy Model в стандартном для EAS виде, значение функции get_core_class были внесены в виде констант, значение которых взято исходя из архитектуры данного тестового стенда (exynos 8890)[10].

Далее для получения информации о загрузке ЦП используется вызов функции из fair.c boosted_cpu_util(). Используется именно эта функция, так как необходимо получать информацию о загруженности от EAS и она имеется в изначальном проекте для тестового стенда.

[1] https://github.com/makar-pelogeiko/herolte_Eas_Idle_modification/tree/Q-stable_idle_modified

[2] https://github.com/jcadduono/lazyflasher/tree/no-verity-opt-encrypt

[3] https://github.com/makar-pelogeiko/Android-idle-state-course-work/tree/course_work_3

Далее для ускорения работы функции выбора состояния простоя сразу выбирается более глубокий сон в следующих случаях:

1. Обрабатывается ядро из класса производительности Little и нагрузка менее 20%.

2. Обрабатывается ядро из класса производительности Middle и нагрузка менее 40%.

3. Обрабатывается ядро из класса производительности Big и нагрузка менее 60%.

   В случае, если таким образом выбрать состояние простоя не удалось, то происходит оценка предполагаемого времени простоя и выбор соответствующего состояния в соответствии с логикой регулятора menu.

5.3 Выбор инструментария тестирования

Для проведения тестирования требовалось определится с инструментами, которые позволяли бы написать и запустить тесты, имитирующие взаимодействие пользователя с GUI на смартфоне. Следует отметить, что необходимо создать условия максимально приближенные к реальному использованию, но при этом воспроизводимые. Для достижения данной цели используется Android Debug Bridge (ADB) - инструмент для взаимодействия с устройством при помощи командной строки[7]. При помощи ADB производится имитация действий пользователя. тестовые сценарии были реализованы в виде *.bat файлов.

В ходе тестирования запускался скрипт сбора данных, который в памяти смартфона. Для всех доступных ядер собирались данные, содержащие время пребывания данного ядра на определенной частоте и общее время пребывания данного ядра в каждом доступном состоянии простоя и количество вызовов этих состояний. Также в процессе тестирования запускался PowerTutor[8] для получения среднего энергопотребления в данном тесте.

5.4 Сценарии тестирования

Для созданий тестов были выявлены частые сценарии использования смартфона в наше время. Из них приоритет отдавался сценариям, не требующим подключения к сети Интернет ввиду снижения неконтролируемых действий смартфона. Таким образом, получились следующие сценарии тестирования:

1. Заметки;

2. Чтение;

3. Видео;
4. Игра;
5. YouTube;

Каждому сценарию теста соответствует фаил .bat, который необходимо запускать при подключенном смартфоне к компьютеру и с включенным adb, который уже имеет доступ к смартфону.
Общая подготовка к тестированию проходила так:
на смартфоне устанавливался режим блокировки "провести по экрану", выключался wifi, bluetooth и энергосбережение, включался режим полета. Так же яркость экрана понижается до минимально возможного уровня.

1. Сценарий Заметки:
   Подготовка сценария теста:
   - Установить приложение Заметки (Turist).
   - Добавить ярлык заметок в левый верхний угол рабочего стола.
   - Запустить приложение заметок, отключить автосохранение, создать 1 текстовую заметку.
   - Переключить язык на английский.
   - Закрыть приложение заметок, выйти на рабочий стол.
   - Заблокировать смартфон и подключить к ПК в режиме отладки.

   Описание теста:
   - Сброс данных о энергопотреблении.
   - Разблокирование смартфона.
   - Включение приложения заметки.
   - Создание новой заметки (текстового поля).
   - Открытие окна печати.
   - Печать текста 2 способами в цикле и сохранение текста.
   - Выход из приложения.
   - Сбор данных о энергопотреблении.
   - Блокирование смартфона.
2. Сценарий Чтение:
   Подготовка сценария теста:

- Добавить ярлык pdf файла в левый верхний угол (на 2 позиции правее заметок из теста 1) рабочего стола.
- Проверить запуск drive pdf viewer 1 кликом.
- Выйти на рабочий стол.
- Заблокировать смартфон и подключить к ПК в режиме отладки.

Описание теста:

- Сброс данных о энергопотреблении.
- Разблокирование смартфона.
- Открытие pdf файла.
- Имитация чтения (протягивание текста).
- Выход из приложения.
- Сбор данных о энергопотреблении.
- Блокирование смартфона.

3. Сценарий Видео:
   Подготовка сценария теста:

   - Добавить ярлык файла mp4 в левый верхний угол (на 2 позиции ниже заметок из теста 1) рабочего стола.
   - Проверить запуск видеоплеера 1 кликом.
   - Выйти на рабочий стол.
   - Заблокировать смартфон и подключить к ПК в режиме отладки.

   Описание теста:

   - Сброс данных о энергопотреблении.
   - Разблокирование смартфона.
   - Открытие видеофайла.
   - Имитация просмотра (бездействие, постановка на паузу, снятие с паузы).
   - Выход из приложения.
   - Сбор данных о энергопотреблении.
   - Блокирование смартфона.

4. Сценарий Игра:
   Подготовка сценария теста:

   - Установить приложение Missiles.

- Добавить ярлык приложения в левый верхний угол рабочего стола (ниже приложения из теста 2).
- Запустить приложение, пройти 2 цикла игры.
- Закрыть приложение, выйти на рабочий стол.
- Заблокировать смартфон и подключить к ПК в режиме отладки.

Описание теста:

- Сброс данных о энергопотреблении.
- Разблокирование смартфона.
- Включение приложения Missiles.
- Имитация игры (запуск новых циклов игры, процесс игры).
- Выход из приложения.
- Сбор данных о энергопотреблении.
- Блокирование смартфона.

5. сценарий YouTube:
   Подготовка сценария теста:

   - установить приложение YouTube.
   - Добавить ярлык файла YouTube-test.html на в левый нижний угол рабочего стола.
   - Открыть файл, кликнуть по ссылке.
   - Закрыть все приложения, выйти на рабочий стол.
   - Заблокировать смартфон и подключить к ПК в режиме отладки.

   Описание теста:

   - Сброс данных о энергопотреблении.
   - Разблокирование смартфона.
   - Включение приложения YouTube по средством открытия файла YouTube-test.html и перехода по ссылке.
   - Имитация просмотра (проверка времени видео).
   - Выход из приложения.
   - Сбор данных о энергопотреблении.
   - Блокирование смартфона.

5.5 Сбор данных

Каждый сценарий тестирование исполнялся 4 раза для обеих версий Android ядер. Для автоматизации представления был написан парсер на Python[4]. Данные тестирования собраны в директории Results[5].

5.6 Анализ результатов

При изучении файла power_profile.xml[9], полученного из тестового стенда выяснилось, что производитель не указал константы энергопотребления. Поэтому было принято решение оценивать энергопотрбление исходя из данных от PowerTutor[8]. На графике показаны медианные значения средних энергопотреблений и стандартное отклонение для каждого тестового сценария. Желтым цветом показаны значения для модифицированного алгоритма состояния простоя, синим – для изначального.

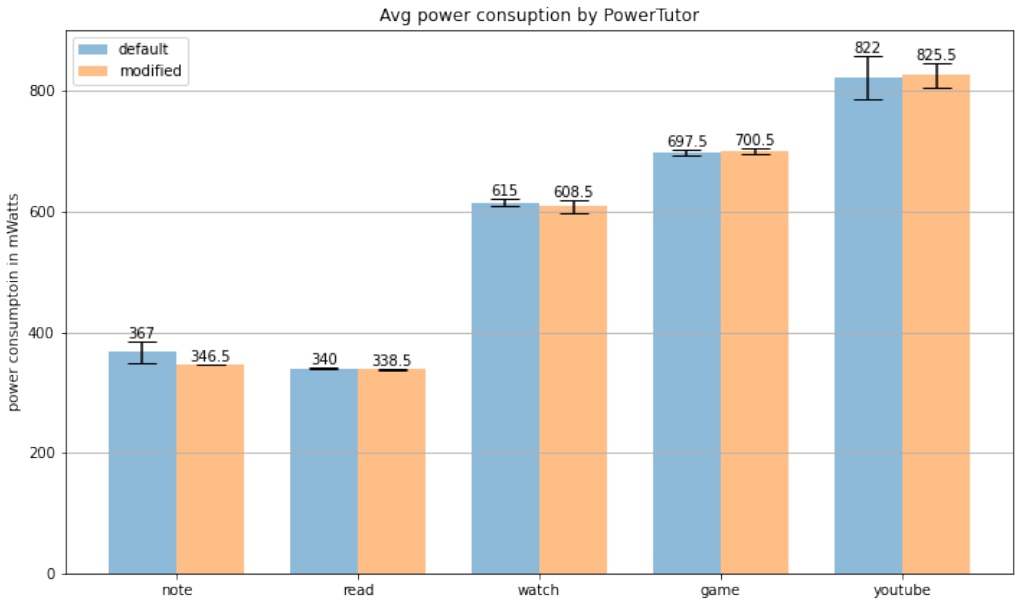

Рис. 2: Среднее потребление в мВаттах.

Далее представлены результаты для каждого сценария тестирования.

В данном тестовом стенде имеются 2 состояния простоя: state 0 и state 1. И в процессе тестирования было определено для отдельных ядер,

[4] https://github.com/makar-pelogeiko/Android-idle-state-course-work/tree/course_work_3/PythonParser

[5] https://studentspburu-my.sharepoint.com/:f:/g/personal/st076963_student_spbu_ru/EmjzqUbRW6pEo67n0FCLh6EBfDgs3Fb8HX4z-PrRPAMjyg?e=My8Ls0

сколько времени проведено в каждом состоянии простоя в микросекундах. Поэтому на графиках для каждого ядра представлены медианные значения времени и стандартное отклонение, которое данное ядро провело в state 0 и state 1 состоянии простоя. Так же были собраны данные для каждого ядра о времени проведенном на определенной частоте в микросекундах. Медианные значения времени для каждой частоты и стандартное отклонение тоже представлены. На всех последующих графиках аналогично предыдущему желтым цветом показаны значения для модифицированного алгоритма состояния простоя, синим – для изначального.

Сравнение времени простоя в сценарий заметки:

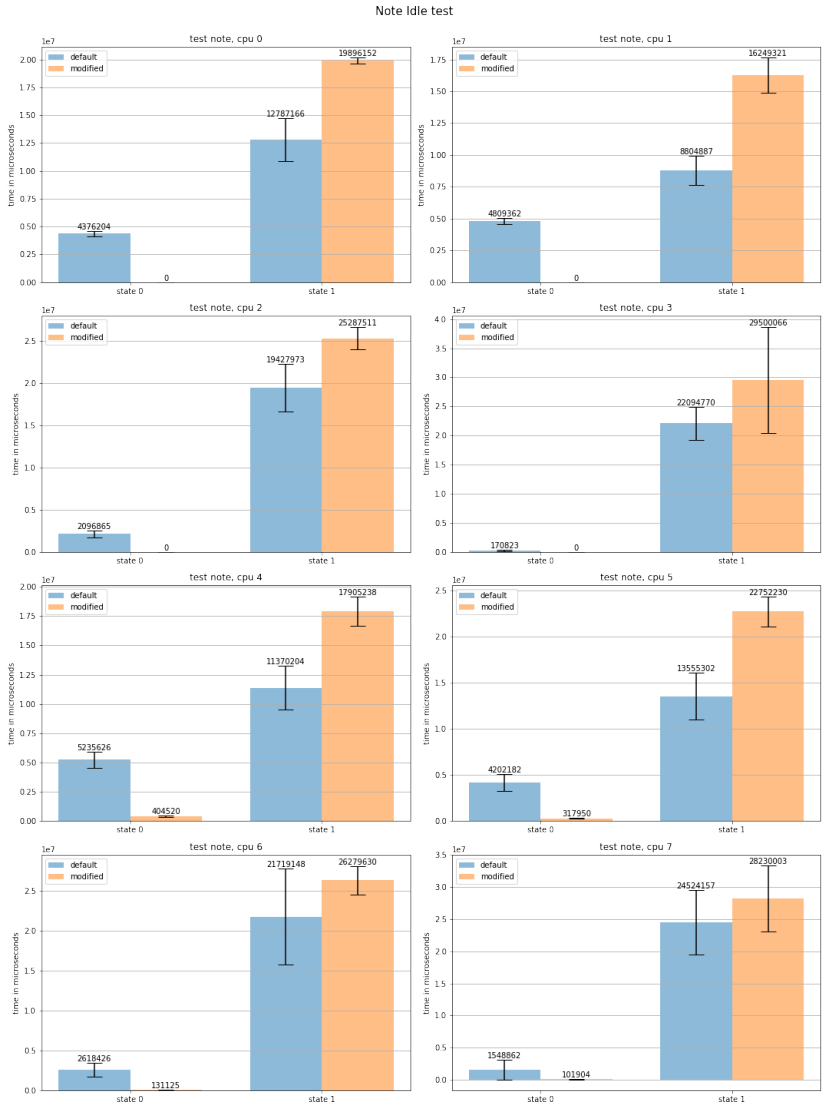

Рис. 3: Время простоя ЦП в микросекундах для сценария заметки.

Сравнение времени, которое ядра ЦП провели на определенных частотах, в сценарий заметки:

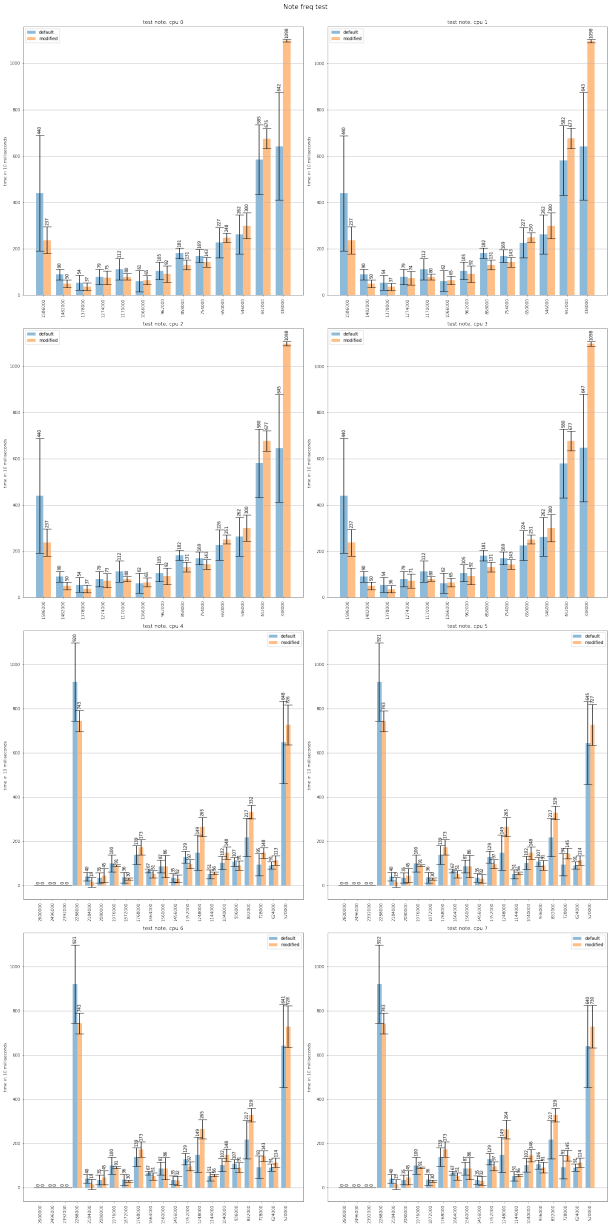

Рис. 4: Время, проведенное на частоте, в 10 милисек., сценарий: заметки.

Сравнение времени простоя в сценарий чтение:

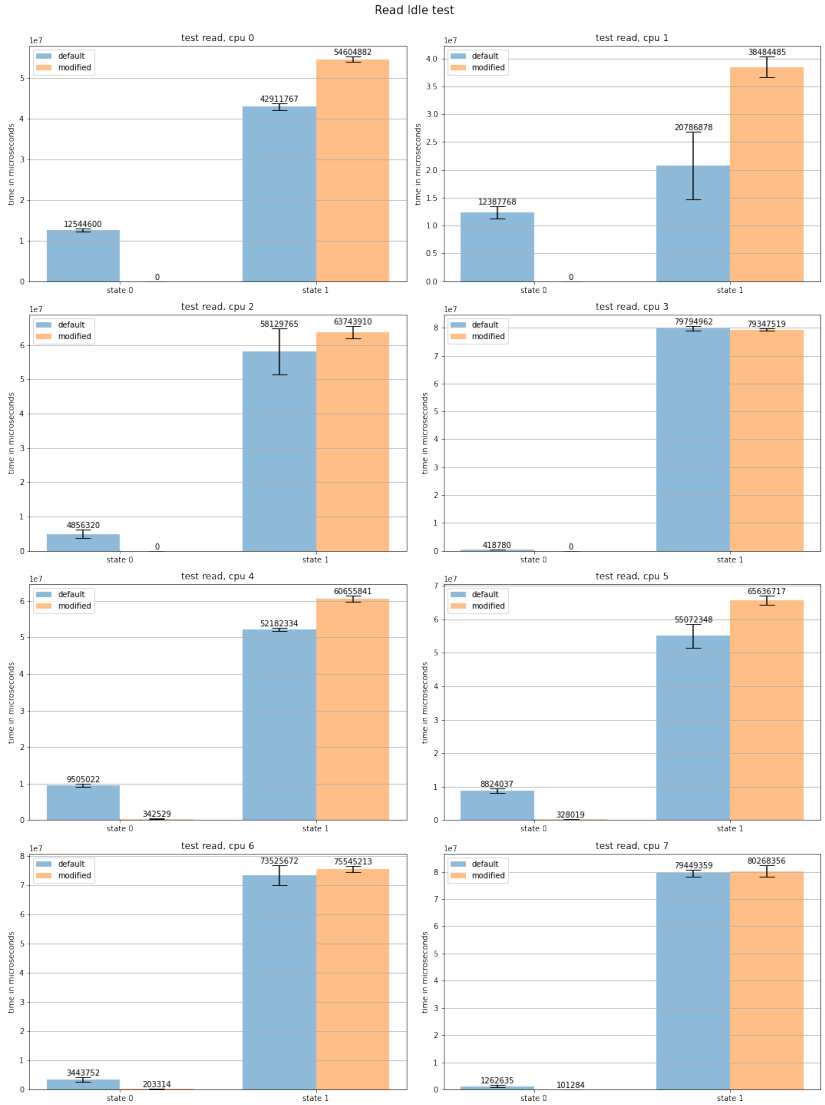

Рис. 5: Время простоя ЦП в микросекундах для сценария чтение.

Сравнение времени, которое ядра ЦП провели на определенных частотах, в сценарий чтение:

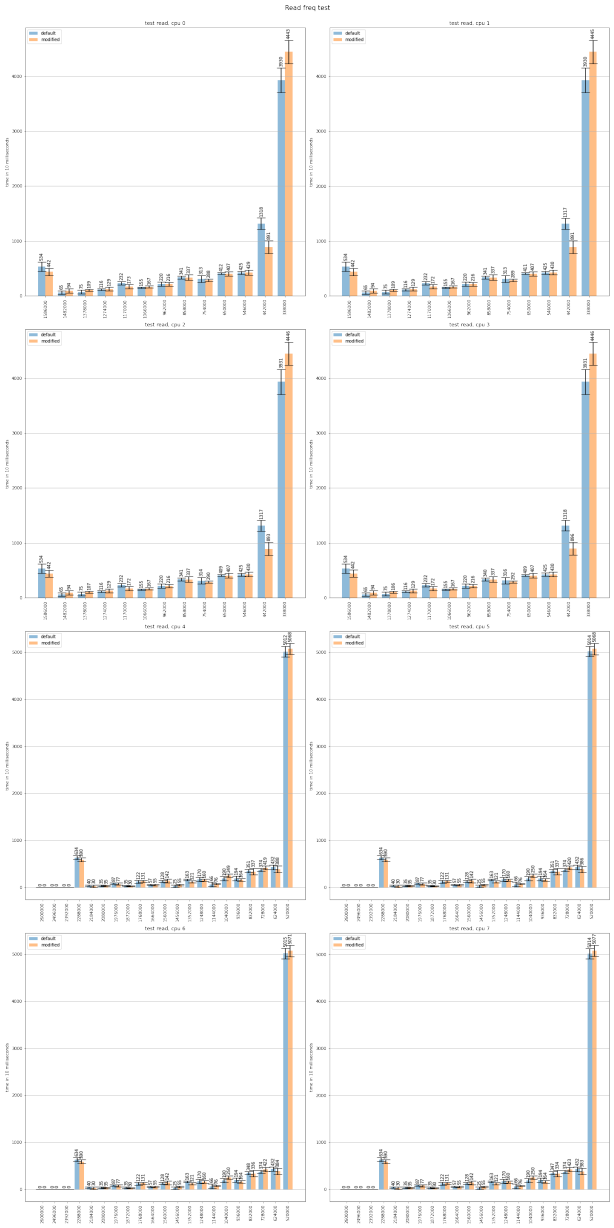

Рис. 6: Время, проведенное на частоте, в 10 милисек., сценарий: чтение.

Сравнение времени простоя в сценарий видео:

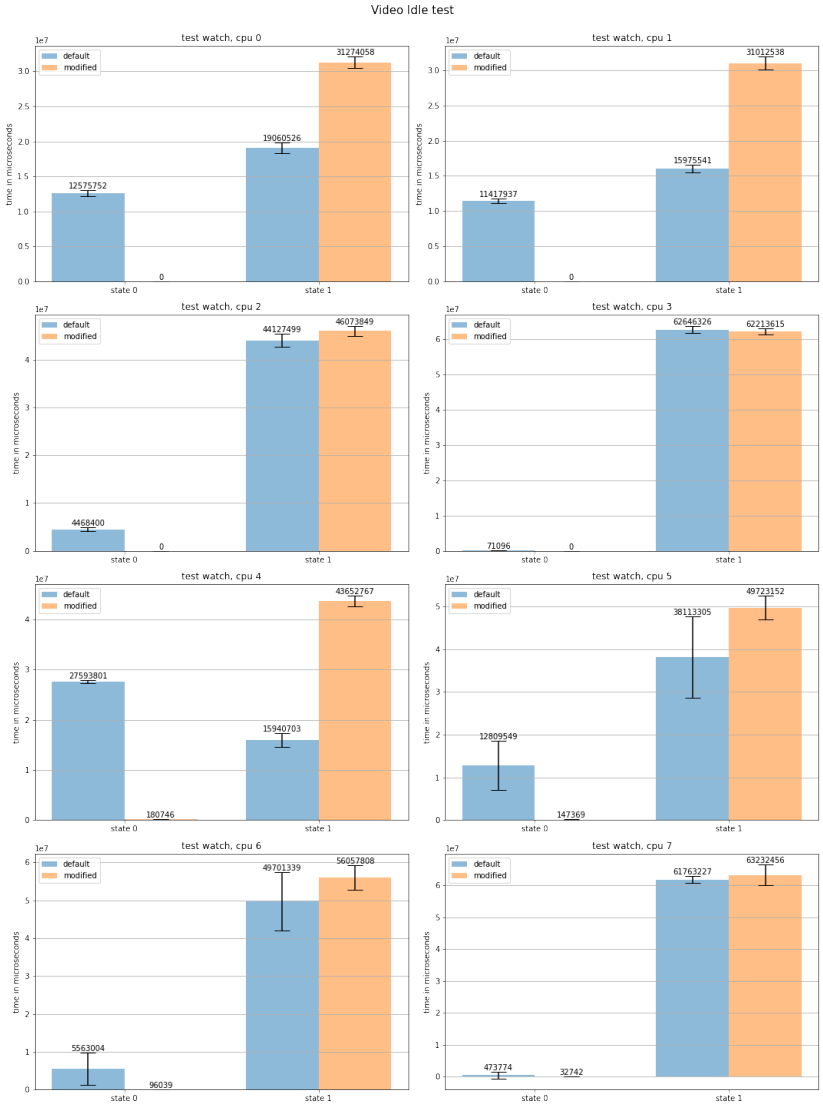

Рис. 7: Время простоя ЦП в микросекундах для сценария видео.

Сравнение времени, которое ядра ЦП провели на определенных частотах, в сценарий видео:

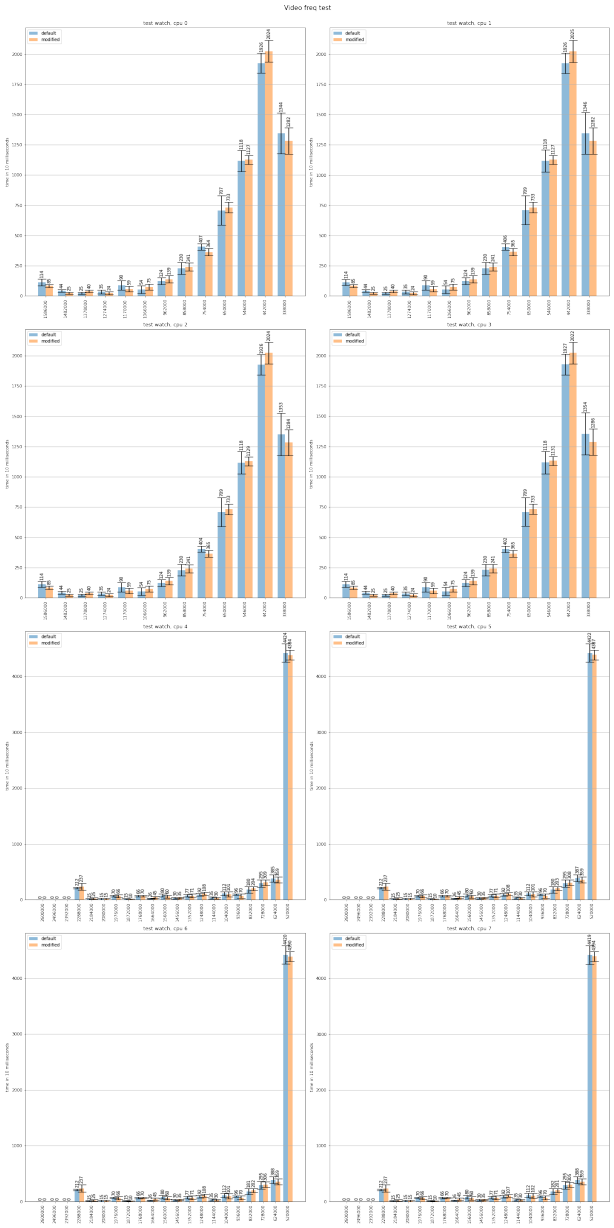

Рис. 8: Время, проведенное на частоте, в 10 милисек., сценарий: видео.

Сравнение времени простоя в сценарий игра:

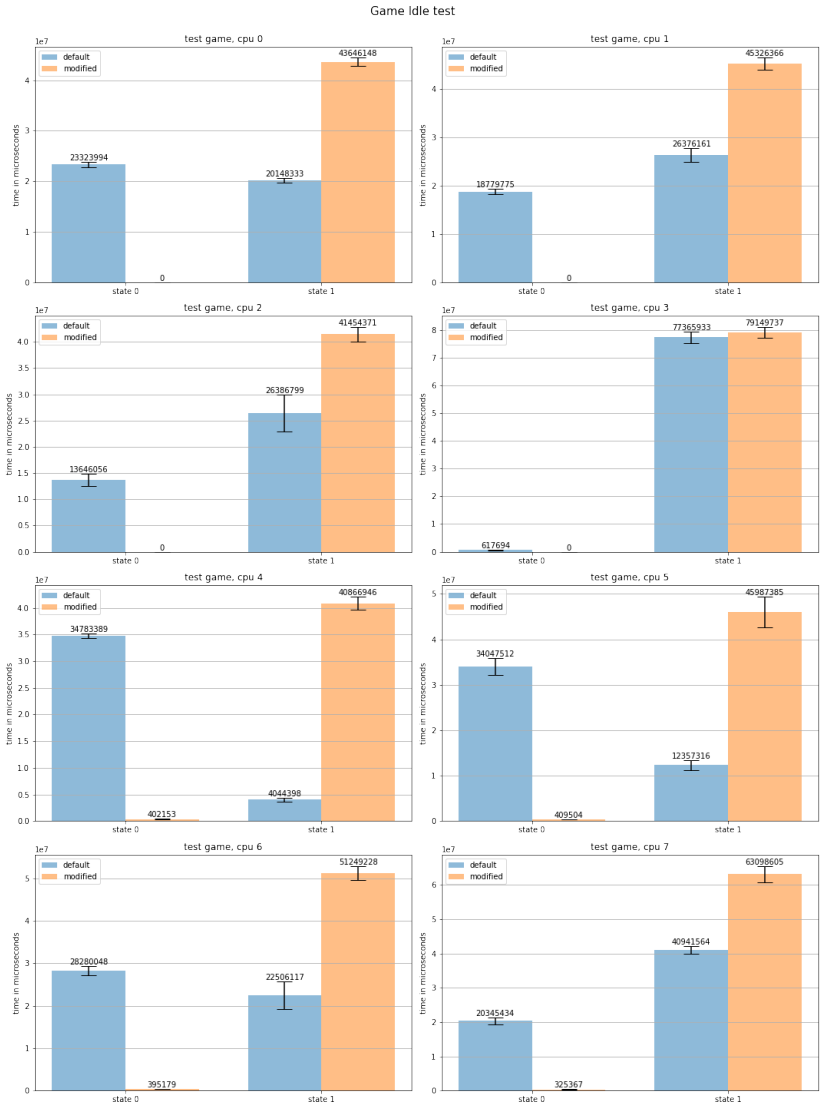

Рис. 9: Время простоя ЦП в микросекундах для сценария игра.

Сравнение времени, которое ядра ЦП провели на определенных частотах, в сценарий игра:

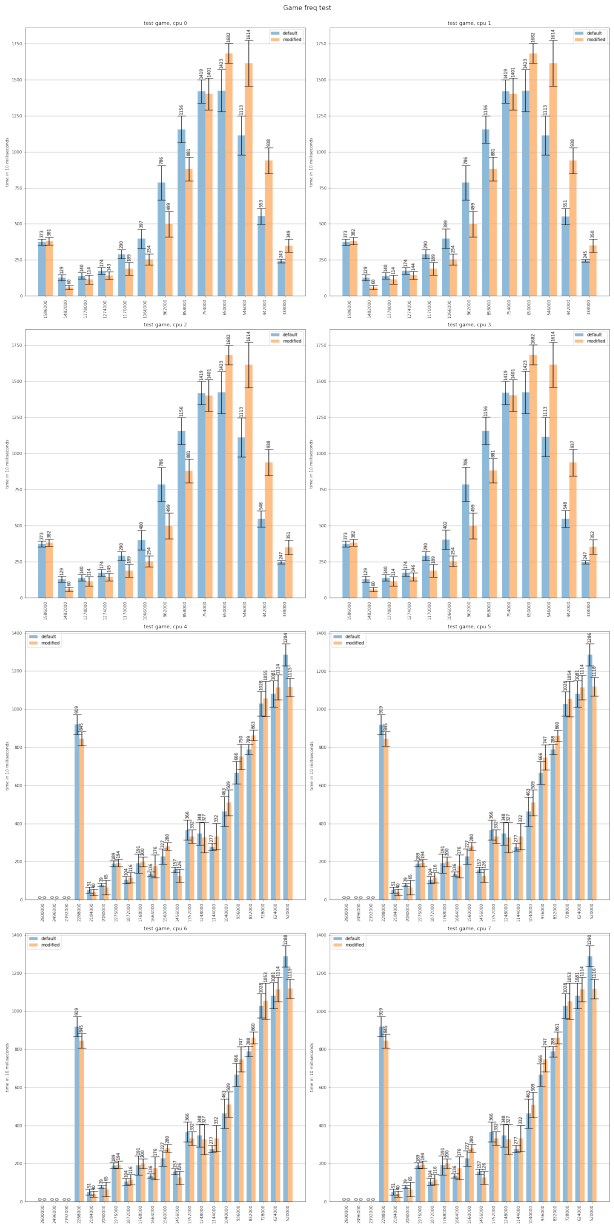

Рис. 10: Время, проведенное на частоте, в 10 милисек., сценарий: игра.

Сравнение времени простоя в сценарий YouTube:

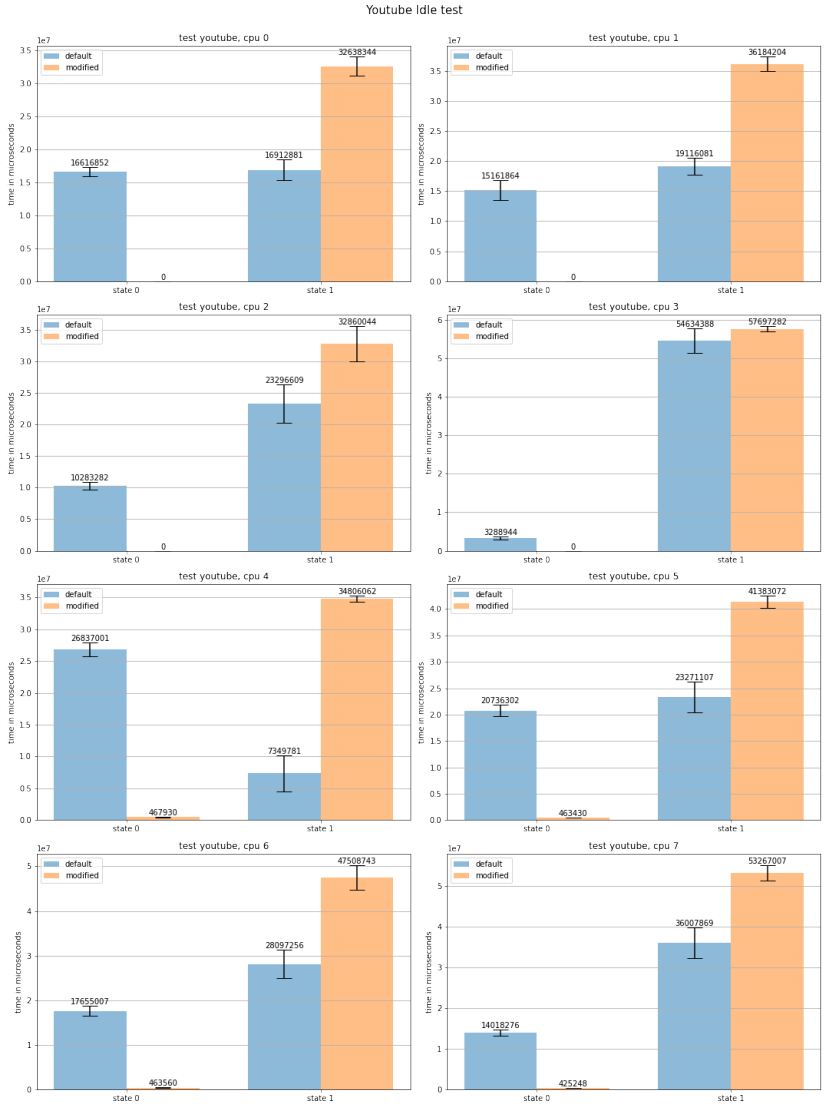

Рис. 11: Время простоя ЦП в микросекундах для сценария YouTube.

Сравнение времени, которое ядра ЦП провели на определенных частотах, в сценарий YouTube:

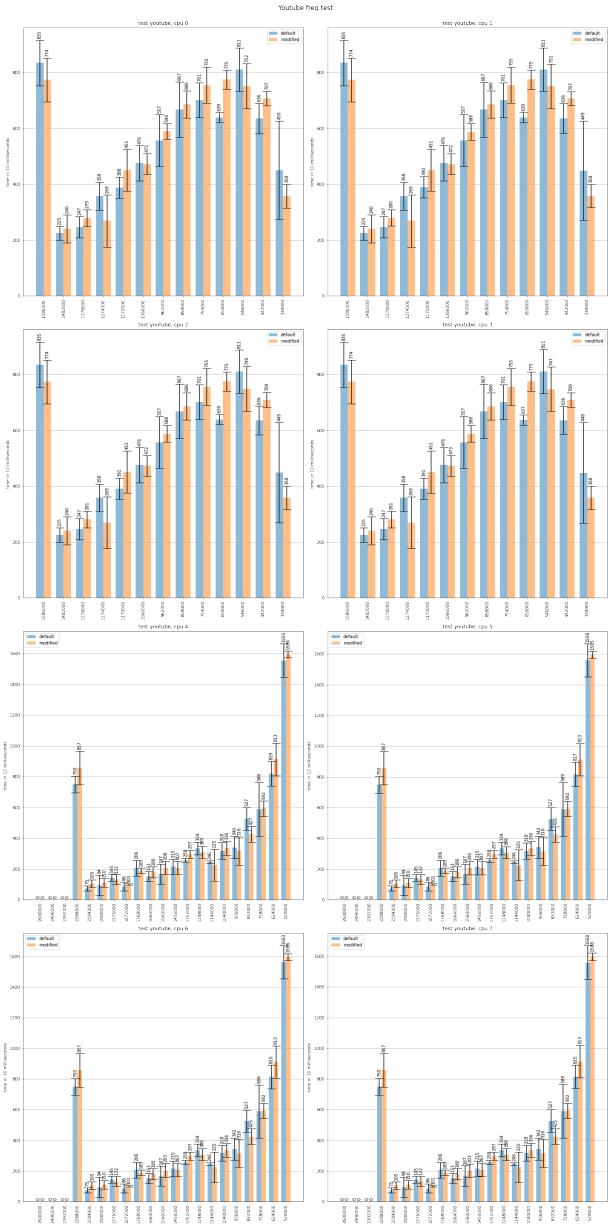

Рис. 12: Время, проведенное на частоте, в 10 милисек., сценарий: YouTube.

Так же было проведено 4 замера производительности для каждой полученной системы при помощи приложения Geekbench 5[11]. На графике представлены медианные значения и отмечен диапазон минимальных и максимальных значений.

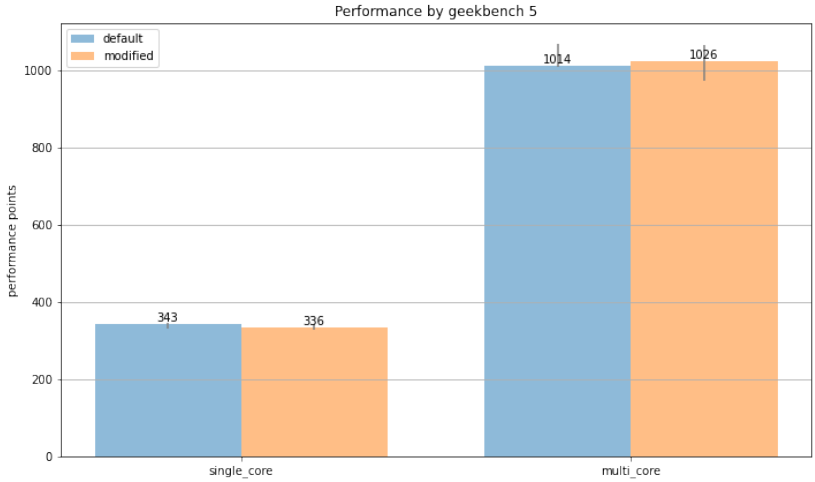

Рис. 13: Производительность ЦП.

Исходя из полученных данных видно, что графики для модифицированного и исходного алгоритмов практически одинаковые, но предсказуемо модифицированный алгоритм показал более низкие результаты.

6 Вывод

Исходя из полученных результатов, можно сказать следующее: получившийся регулятор и регулятор menu показали близкие друг к другу результаты. У системы с модифицированным регулятором состояния простоя энергопотребление немного ниже чем у системы с регулятором menu при использовании смартфона с не интенсивными нагрузками. Однако при использовании смартфона с более высокой нагрузкой, например, запуска мобильной игры, система с регулятором menu показывает немного меньшее энергопотребление, относительно системы с модифицированным регулятором. Это можно объяснить тем, что из-за большого количества раз включения глубокого состояния простоя необходимо тратить энергию на выход из него, кроме того в время активной работы ядра приходится повышать повышать частоту.

Список литературы

[1] CPUIdle Time Management 2018:
https://www.kernel.org/doc/html/latest/admin-guide/pm/cpuidle.html

[2] CPUIdle Time Management 2019:
https://www.kernel.org/doc/html/latest/driver-api/pm/cpuidle.html

[3] Improving idle behavior in tickless systems:
https://lwn.net/Articles/775618/

[4] The cpuidle subsystem:
https://lwn.net/Articles/384146/

[5] The cpuidle subsystem:
https://lwn.net/Articles/384146/

[6] G. Metri, A. Agrawal, R. Peri, M. Brockmeyer and Weisong Shi, "A simplistic way for power profiling of mobile devices" 2012:
https://ieeexplore.ieee.org/abstract/document/6471020

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

[8] PowerTutor:
http://ziyang.eecs.umich.edu/projects/powertutor/

[9] Power profile:
https://source.android.com/devices/tech/power/values

[10] Exynos 8890:
https://en.wikichip.org/wiki/samsung/exynos/8890

[11] Geekbench 5:
https://www.geekbench.com/