Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20Б.10–мм

Волков Александр Дмитриевич

Энергосбережение в Android OS путём динамического изменения частоты процессора

Отчёт по учебной практике
в форме «Решение»

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

Санкт-Петербург
2022

Оглавление

1. Введение

На данный момент мобильные устройства используются повсеместно, играя значительную роль в жизни человека. Большинство из них работает на базе операционной системы Android, значительно опережающей по популярности своего основного конкурента — iOS [6]. К мобильным устройствам предъявляют все больше требований, что приводит к постоянному росту вычислительных мощностей процессоров. Это неизбежно влечет за собой сильное увеличение энергопотребления, при этом возможности аккумуляторов не растут столь же быстро.

Возникает задача оптимизации работы устройства для продления времени работы устройства без аппаратных изменений. Один из способов такой оптимизации — динамическое изменение напряжения частоты процессора. Такой подход оправдан тем, что, несмотря на высокую мощность используемых процессоров, многим приложениям не требуется столько ресурсов. Для изменения частоты в системе Android существует модуль CPUfreq subsystem [4], предоставляющий интерфейс для DVFS (Dynamic Voltage and Frequency Scaling) регуляторов.

На текущий момент существует несколько регуляторов, включенных в большинстве реализаций системы Android, однако не доказано, что их работа является оптимальной. Также существует множество алгоритмов, предложенных в различных исследованиях на данную тему. В работах [9, 11] был предложен DVFS-алгоритм, основанный на стохастической аппроксимации со случайными направлениями (SPSA) [13].

Но алгоритм был реализован в соответствии с версией алгоритма SPSA с одним измерением, что может негативно сказываться на результатах работы. Поэтому было принято решение создать DVFS-регулятор на основе версии алгоритма SPSA с двумя измерениями и изучить его свойства.

2. Постановка задачи

Целью работы является создание DVFS-регулятора на основе алгоритма SPSA с двумя измерениями и изучение его характеристик. Для её выполнения были поставлены следующие задачи.

1. Провести обзор существующих на данный момент DVFS-регуляторов.
2. Провести обзор алгоритмов SPSA.
3. На основе уже существующего регулятора [9, 11] реализовать базовую версию регулятора, основанного на алгоритме SPSA с двумя измерениями.
4. Оптимизировать настраиваемые параметры регулятора.
5. Провести оценку эффективности работы алгоритма и сравнение с другими регуляторами.

3. Обзор

3.1. Динамическое изменение частоты

Далее будут рассмотрены регуляторы частоты для устройств на базе операционной системы Android.

3.1.1. Стандартные регуляторы

Стандартная версия системы содержит [3] следующие регуляторы.

- Performance — устанавливает частоту на максимальное значение.
- Powersave — устанавливает частоту на минимальное значение.
- Userspace — разрешает пользоваетелю или любой другой программе, запущенной от имени root, устанавливать частоту на специфическое значение, делая файл scaling_setspeed доступным в каталоге CPU-device.
- Ondemand — устанавливает текущую частоту в зависимости от нагрузки на систему. Системные файлы, определяющие параметры работы регулятора:
  - sampling_rate — как часто нужно производить измерения и обновлять частоту. Измеряется в микросекундах;
  - sampling_rate_min — минимальная частота изменений;
  - up_threshold — показывает уровень загруженности системы, при превышении которого необходимо повысить частоту. Измеряется в процентах;
  - ignore_nice_load — если равен 1, то при рассчете нагрузки на систему не будут учитываться процессы, запущенные с модификатором «nice». Полезен, когда на устройстве проводятся вычисления, время выполнения которых неважно и которые можно не учитывать при поиске оптимальной частоты. Значение по умолчанию — 0;

  - sampling_down_factor — определяет, как часто будут проводиться измерения при работе на максимальной частоте. По умолчанию равен 1, но при больших значениях (до 100) выполняет роль множителя для периода проверок. Это позволяет снизить дополнительные расходы при максимальной загруженности;
  - powersave_bias — принимает значения от 0 до 1000. Он определяет процент (умноженный на 10) значения, которое будет отнято от вычисленной частоты. К примеру, когда он установлен на 100 — 10%, когда ondemand требует 1000 МГц, будет выставлена частота 900 МГц. По умолчанию установлен на 0;

- Conservative — как и ondemand, меняет частоту в зависимости от текущей нагрузки. Отличие заключается в том, что conservative делает это плавно, без резких скачков на высокие частоты. Такое поведение предпочтительней на устройствах, работающих от батареи, то есть на всех мобильных. Имеет те же параметры, что и ondemand, помимо своих собственных:
  - freq_step — процент (от максимальной частоты), на который следует изменять частоту;
  - down_threshold — аналогичен параметру up_threshold, но работает в обратную сторону (нижняя граница, при достижении которой надо понизить частоту);
  - sampling_down_factor — обладает той же функциональностью, что и sampling_down_factor у ondemand, но здесь он определяет период проверок для снижения частоты во время работы на любой частоте;

- Interactive — разработан для чувствительных к задержкам задач. Этот регулятор устанавливает скорость процессора в зависимости от нагрузки на систему, подобно ondemand и conservative. Тем не менее, interactive активнее реагирует на новые задачи. Регулятор

не дожидается окончания периода для нового измерения, а реагирует на каждый выход из состояния покоя. Если регулятор обнаруживает высокую нагрузку, то сразу же повышает частоту до максимальной. Если нагрузка не требует повышения частоты до максимальной, то регулятор сравнивает текущюю нагрузку с нагрузкой с последнего изменения частоты и берет максимальное значение из этих двух. На основне него он получает новую целевую частоту. Параметры этого регулятора:

- min_sample_time — минимальное время, которое необходимо провести на текущей частоте перед снижением. Необходимо для того, чтобы убедиться, что регулятор зарегестрировал достаточное количество данных для последующей корректной работы. Значение по умолчанию — 80000 мкс;
- hispeed_freq — высокая частота, на которую необходимо переключиться, когда нагрузка достигает go_hispeed_load. Если нагрузка остается высокой в течение времени, определенного в above_hispeed_delay, то частота может быть повышена. Значение по умолчанию — максимальная допустимая частота;
- go_hispeed_load — нагрузка, при которой надо сразу же переключиться на высокую частоту. Значение по умолчанию — 85%;
- above_hispeed_delay — как только частота установлена на hispeed_freq, необходимо ожидать в течение этого временного промежудтка прежде чем повышать частоту дальше. Значение по умолчанию — 20000 мкс;
- timer_rate — с этой частотой надо изменять частоту, когда система не находится в состоянии покоя. Значение по умолчанию — 20000 мкс;
- input_boost — если не равн нулю, нужно увеличить частоту всех ядер до hispeed_freq во время работы с экраном. Значе-

ние по умолчанию — 0;

- boost — если не равне нулю, необходимо увеличить частоту всех ядер до hispeed_freq, пока параметр опять не станет нулевым. Если равен нулю, разрешает частотам ядер опускаться ниже hispeed_freq в соответствии с нагрузкой;
- boostpulse — поднимает частоты всех ядер до hispeed_freq на время min_sample_time, после чего все частоты могут опуститься ниже hispeed_freq в соответствие с нагрузкой;

3.1.2. Сторонние регуляторы

В работе [12] описываются 12 исследований, посвященных оригинальным алгоритмам DVFS. В этих работах авторы использую различные подходы, такие как машинное обучение, адаптивные алгоритмы и т.д. Но, к сожалению, во всех случая авторы не предоставляют готовый регулятор на основе своего алгоритма, так что единственный вариант использовать его — самостоятельно реализовать DVFS-регулятор на основе предложенного алгоритма, что не представляется возможным.

Помимо существующих алгоритмов, в работе [12] были рассмотрены и уже готовые решения. Из них стоит выделить schedutil, так как прочие представленные регуляторы являются его модификациями. Он показывает хорошие результаты в плане энергопотребления и производительности, но сравнивать его с регуляторами из 3.1.1 и регулятором, представленным в данной работе, некорректно, так как schedutil является частью более крупной системы - планировщика задач EAS.

3.2. Алгоритмы SPSA

Из работ [11], [13] известно, что если $F_t(x, w)$ — случайная функция от дискретного времени t, параметра x и случайного вектора w, $f_t(x) = \mathbb{E}_w F_t(x, w)$ — функционал среднего риска, $\theta_t = \arg\min_x f_t(x)$ — точка минимума $f_t(x)$, то для построения последовательности оценок точки минимума функционала $f(x)$ можно воспользоваться алгоритмом (Δ_n

принимает значения ± 1 с равной вероятностью):

$$\widehat{\theta}_n = \widehat{\theta}_{n-1} - \alpha\Delta_n \frac{y_n^+ - y_n^0}{\beta}$$

где используются два (v_n^+, v_n^0 - шумы) наблюдения:

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

3.2.1. Версия алгоритма с одним измерением

В работе [11] используется версия алгоритма с одним измерением:

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

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

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

$$F_\tau(f) = 2^{(workload_\tau(f)-\lambda)/2} + \gamma 1.5^{table(f)} \quad (2)$$

$table(f)$ — номер частоты в таблице частот. Получен следующий алгоритм: Freq — множество частот, доступных процессору, P$(\cdot)$ — проектор в множество Freq, L$(\cdot)$ — проектор в отрезок [min(Freq), max(Freq)]

1: Выбираем начальное значение оценки: $\widehat{f_0}$
2: Генерируем очередной элемент случайной последовательности чисел, равных ± 1 с одинаковой вероятностью — Δ_n
3: Возмущаем текущую оптимальную оценку: $f_n = P(\widehat{f}_{n-1} + \beta\Delta_n)$,
4: Запускаем систему на f_n
5: Получаем зашумленное наблюдение: $y_n = F_\tau(\widehat{f}_{n-1} + \beta\Delta_n) + v_n$
6: Обновляем текущую оптимальную оценку: $\widehat{f_n} = L(\widehat{f}_{n-1} - \alpha\Delta_n \frac{y_n}{\beta})$
7: Переходим к шагу 2

3.2.2. Версия алгоритма с двумя измерениями

Здесь же предлагается использовать версию с двумя измерениями:

$$\widehat{\theta_n} = \widehat{\theta}_{n-1} - \alpha_n \Delta_n \frac{y_n^+ - y_n^0}{\beta_n}$$

Метрика и модель остаются прежними — (1) и (2). Итоговый алгоритм:

1: Выбираем некоторое начальное значение $\widehat{f_0}$, например $max(Freq)$
2: Запускаем систему на $P(\widehat{f_n})$
3: Получаем зашумленное наблюдение $y_n^0 = F_\tau(\widehat{f_n}) + v_n^0$
4: Генерируем новое значение Δ_n, равное ± 1 с одинаковыми вероятностями
5: Возмущаем текущую оптимальную оценку $f_n = P(\widehat{f_n} + \beta\Delta_n)$
6: Запускаем систему на f_n
7: Получаем зашумленное наблюдение $y_n^+ = F_\tau(\widehat{f_n} + \beta\Delta_n) + v_n^+$
8: Обновляем текущую оптимальную оценку $\widehat{f}_{n+1} = L(\widehat{f_0} - \alpha\Delta_n \frac{y_n^+ - y_n^0}{\beta})$
9: Переходим к шагу 2

Таким образом, отличаются шаги 2, 3 и 8.

3.3. Технические средства

3.3.1. Xiaomi Redmi Note 8 Pro

Для экспериментов был выбран смартфон Xiaomi Redmi Note 8 Pro, обладающий следующими характеристиками:

- 6 гигабайт оперативной памяти;
- 64 гигабайта основной памяти;
- ОС Android 10;
- 8-ми ядерный процессор Mediatek Helio G90T;

Выбор был обусловлен как объективными причинами (современное аппаратное обеспечение и актуальная операционная система, популяр-

ность и т.д.), подробнее изложенными в [11, 12], так и тем, что смартфон уже был в значительной степени готов к работе.

3.3.2. Программное обеспечение

На смартфоне уже была установлена прошивка POSP [8], разработанная для данной модели смартфона, и ядро AgentFabulous begonia kernel project [1]. Также на смартфон была установлена утилита восстановления TWRP [10] и Magisk [5], предоставляющий права суперпользователя. Подробнее процесс установки изложен в работах [11, 12].

4. Ход работы

4.1. Исправление ошибок

Перед началом работы были исправлены следующие ошибки:

- Изначально регулятор spsa было невозможно выбрать в качестве текущего. Проблема была решена после замены ondemand_copy на spsa в некоторых файлах (ondemand_copy — временное название, использовавшееся при разработке регялтора spsa).
- Были возвращены удаленные по ошибке строки в файле cpufreq_spsa.c.

4.2. Новый регулятор

Была добавлена модификация регулятора spsa под названием spsa2, реализующая алгоритм SPSA с двумя измерениями.

Ранее работа происходила следующим образом: при вызове функции od_update вычислялась текущая нагрузка на процессор и на основе этого получалась текущая оптимальная оценка, которая записывалась в поле cur_estimation структуры, отвечающей за хранение информации регулятора. Далее вычислялось значение freq_next, получаемое как сумма cur_estimation и слагаемого, отвечающего за случайные возмущения. После этого вызывалась функция __cpufreq_driver_target, устанавливавшая частоту на значение freq_next.

Теперь были добавлены переменные для хранения о состоянии системы на момент предыдущего вызова функции od_update:

- spsa_phase — равно 0, если в данный момент измерения проводятся на частоте cur_estimation и зашумленное измерение необходимо записать в переменную old_model. Равно 1, если в данный момент измерения проводятся на возмущенной оптимальной оценке;
- old_model — в этой переменной хранится зашумленное измерение, проведенное на частоте cur_estimation;

Если spsa_phase равно нулю, то после проведения измерений текущая оптимальная оценка не изменяется, а данные просто записываются в переменную old_model. Если же spsa_phase равно 1, то вычисляется новая оптимальная оценка.

4.3. Модификации

Были протестированы различные вариации регулятора на основе алгоритма SPSA с двумя измерениями. Поиск параметров производился с помощью моделирования работы системы при различных значениях переменных.

4.3.1. SPSA с двумя измерениями с одной фазой

Частота, определяемая регулятором, задается на сравнительно большой промежуток времени (по сравнению с периодом работы планировщика). В связи с этим возника идея получать второе измерение на основе первого, а не проводить ещё один замер. Хотя такой подход и является более простым с технической точки зрения, он не соответсвует изначальной идее SPSA2, так как оба измерения должны быть зашумленными.

В данном случае второе наблюдение получается на основе тех же данных, что и первое. Т.е. после получения возмущенной частоты нагрузка для нее рассчитывается как $current\_load * \frac{current\_frequency}{disturbed\_frequency}$. На основе этого вычисляется второе измерение и далее работа алгоритма не отличается от версии с двумя фазами.

Была реализована соотвествующая версия регулятора cpufreq_spsa2.c.

4.3.2. SPSA с двумя измерениями с двумя фазами

Данный алгоритм ничем не отличается от "обычной" версии, т.е. здесь используются два зашумленных измерения.

При тестировании найденный для этого алгоритма параметров были проверены различиные версии регулятора, отличающиеся использова-

нием или неиспользованием функции, находящей ближайшее к вычисленной частоте значение из возможных частот для текущего кластера.

4.4. Моделирование

Научный руководитель реализовал несколько моделей, облегчающих поиск оптимальных значений параметров, на языке C#. Их работа основана на предсказании частоты, на которой в определенной момент времени должен работать процессор. Для оценки того, насколько модель эффективна, необходимы замеры, полученные в ходе реальной работы устройства. Оптимальные параметры выбирались на основании предполагаемого энергопотребления и производительности. Для оценки производительности использовалось отношение числа раз, когда модель предлагала частоту выше, чем реальный регулятор, к числу раз, когда модель предлагала частоту ниже.

В код регулятора был добавлен модуль, записывающий в память устройства временную метку, кластер, текущую частоту и максимальную нагрузку на ядра этого кластера за последний период, каждый раз, когда вызывается основная функция регулятора, отвечающая за изменение частоты. На основании этих данных и производилось моделирование системы.

5. Тестирование

При тестировании алгоритма применяются результаты, полученные в работе [12]. Новые тесты унаследовали от уже существующих инструмент для тестирования и основные команды. Также была добавлена возможность запускать тесты с помощью вызова соответсвующих функций.

5.1. Инструмент для тестирования

Для тестирования будет использоватья утилита Monkeyrunner [7]. Monkeyrunner предоставляет API для написания программ на языке Python для работы с Android устройством. Использует Jython для запуска скриптов. Подробнее причины выбора именно этого инструмента и сравнение с аналогами изложены в работе [12], здесь же отметим:

- Отсутствие требований к приложению, на котором будет проводиться тестирование, т.е. для тестов необходим лишь .apk файл.
- Возможность работать с adb (Android Debug Bridge) [2].

5.2. Тестовые случаи

5.2.1. Существующие

- Camera test — установка приложения для съемки Open Camera, если оно не было установлено до этого, запуск камеры смартфона, съемка видео в течение 15 минут.
- Typing test — установка приложения для заметок Notes, 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, 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.

- Video test — установка приложения VLC player for Android 3 и воспроизведение 15-минутного видео.

5.2.2. Новые

- PDF test — запуск приложения Foxit PDF Editor и имитация чтения достаточно большого pdf документа в течение 15 минут.

- Youtube test — проигрывание видео на сайте youtube.com с помощью бразузера Firefox в качестве 480p в течение 15 минут;

Новые тесты были выбраны по двум причинам:

- в существующих тестах полностью исключается использование интернета, в то время как значительная часть приложений использует его, так что влияние использования сетевого подключения для передачи данных на работу также необходимо исследовать (youtube test);

- в существующих тестах нагрузка относительно стабильная, а в новых (pdf test, youtube test) она неравномерная, что сделает оценку работы регуляторов более точной;

5.3. Результаты

В результате экспериментов были получены следующие данные. В первой таблице представлены оценки производительности смартфона при работе с разными регуляторами (использованный бенчмарк — AnTuTu). Во второй таблице — суммарное потребление электроэнергии в тестах с различными регуляторами (каждый тест — 15 минут).

Таким образом, регулятор SPSA2 с двумя фазами с найденными параметрами продемонстрировал уменьшение энергопотребления и про-

Governor	Total	CPU	GPU	Memory	UX
OnDemand	294576	83835	85450	45805	79486
Interactive	302304	87854	86427	47101	80922
SPSA	305935	88149	87668	46477	83641
SPSA2 (с одной фазой)	313528	90606	89004	49380	84538
SPSA2 (с двумя фазами)	233199	57636	70437	41357	63769

Таблица 1: AnTuTu benchmark

Test	OnDemand	Interactive	SPSA	SPSA2 (с двумя фазами)	SPSA2 (с одной фазой)
Camera test	76	44	61	59	66
Typing test	75	64	69	61	69
Trial Xtreme 3	49	40	41	43	49
Flappy bird	32	39	36	31	43
Video test	31	51	32	42	41
Pdf test	75	66	46	45	52
Youtube test	52	43	50	60	61

Таблица 2: Энергопотребление (в мАч)

изводительности по сравнению с SPSA. Регулятор SPSA2 с одной фазой продемонстрировал рост энергопотребления и снижение производительности, так что его использование с такими параметрами представляется нецелесообразным.

6. Заключение

За осенний семестр было сделано.

- Был проведен обзор DVFS-регуляторов и алгоритмов SPSA.
- Была реализована базовая версия DVFS-регулятора на основе алгоритма SPSA с двумя измерениями.
- Регулятор был частично оптимизирован.
- Был сформирован набор тестов для оценки эффективности регулятора.

В весеннем семестре было сделано.

- Разработаны алгоритмы, имитирующие работу регулятора spsa2.
- Найдены параметры, близкие к оптимальным.
- Проведено сравнение с другими регуляторами.

Ссылка на репозиторий с кодом: github.com/al-volkov/DVFS-for-begonia
Ссылка на репозиторий с тестами: github.com/al-volkov/dvfs_testing

Список литературы

[1] AgentFabulous begonia kernel project. — URL: https://github.com/AgentFabulous/begonia (online; accessed: 2021-12-12).

[2] Android Debug Bridge. — URL: https://developer.android.com/studio/command-line/adb (online; accessed: 2021-12-12).

[3] CPU frequency and voltage scaling code in the Linux kernel. — URL: https://android.googlesource.com/kernel/common/+/a7827a2a60218b25f222b54f77ed38f57aebe08b/Documentation/cpu-freq/governors.txt (online; accessed: 2021-12-12).

[4] The Linux Kernel user's and administrator's guide. — URL: https://www.kernel.org/doc/html/v4.14/admin-guide (online; accessed: 2021-12-12).

[5] Magisk Manager. — URL: https://magiskmanager.com/ (online; accessed: 2021-12-12).

[6] Mobile Operating System Market Share Worldwide. — URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (online; accessed: 2021-12-12).

[7] Monkeyrunner. — URL: https://developer.android.com/studio/test/monkeyrunner (online; accessed: 2021-12-12).

[8] Potato Open Sauce Project. — URL: https://potatoproject.co/ (online; accessed: 2021-12-12).

[9] Stochastic DVFS for begonia kernel. — URL: https://github.com/jackbogdanov/DVFS-for-begonia (online; accessed: 2021-12-12).

[10] Team Win Recovery Project. — URL: https://twrp.me (online; accessed: 2021-12-12).

[11] Богданов Е.А. Использование стохастической оптимизации для регулировки частоты процессора в Android OS. — 2021. —

URL: https://oops.math.spbu.ru/SE/diploma/2021/pi/Bogdanov-Evgenii-report.pdf (online; accessed: 2021-12-12).

[12] Божнюк А.С. Сравнение эффективности алгоритмов DVFS для оптимизации энергопотребления Android устройств. — 2021. — URL: https://oops.math.spbu.ru/SE/YearlyProjects/spring-2021/uchebnye-praktiki/pi-2/Bozhnyuk-report.pdf (online; accessed: 2021-12-12).

[13] Граничин О.Н. Рандомизированные алгоритмы оценивания при нерегулярных помехах // Стохастическая оптимизация в информатике. — 2006. — URL: https://www.gsom.spbu.ru/files/upload/niim/news/2006/301006Granichin1.pdf (online; accessed: 2021-12-12).