Министерство образования и науки Российской Федерации
Московский физико-технический институт
(национальный исследовательский университет)

Физтех-школа радиотехники и компьютерных технологий
Кафедра микропроцессорных технологий в интеллектуальных системах управления

Выпускная квалификационная работа бакалавра

Политика регулирования частот центрального процессора в ядре Linux для приложений реального времени

Автор:
Студент Б01-008 группы
Глаз Роман Сергеевич

Научный руководитель:
Кринов Пётр Сергеевич, к. ф.-м. н.

Москва 2024

Аннотация

Политика регулирования частот центрального процессора в ядре Linux для приложений реального времени

Глаз Роман Сергеевич

На данный момент в мире насчитывается более 4-ёх миллиардов пользователей смартфонов. Несмотря на широкий диапазон возможностей смартфона, производительность, длительное время автономной работы и надежность являются основными критериями его конкурентоспособности.
Преобладающая часть смартфонов использует архитектуру ARM big.LITTLE – архитектура ARM/ARM64, основой которой является использование различных типов ядер – энергоэффективных с низкой производительностью и высокопроизводительных, имеющих большое энергопотребление.
Также одной из основных технологий, используемых процессорами ARM, является Dynamic Voltage and Frequency Scaling (DVFS), благодаря которой удаётся снизить энергопотребление процессора, снижая его частоту работы и напряжение, если нагрузка на процессор небольшая.
Для разработки алгоритмов планировщика операционных систем, который учитывает как наличие ядер разной производительности, так и технологию DVFS, очень удобным инструментом является симулятор микроархитектуры, позволяющий использовать подробную статистику использования различных компонент симулируемой системы. В данной работе используется симулятор Gem5 и Linux в качестве исследуемой операционной системы.
В данной работе:

1. Разработаны компоненты симулятора Gem5, предоставляющие возможность создавать гетерогенные (мультикластерные) архитектуры с возможностью использования ARM Performance Monitor Unit (PMU) счётчиков, связанных с иерархией кешей.

2. Реализованы драйверы ядра Linux (версии 6.1) для платформ Gem5, предоставляющие поддержку DVFS как для процессоров (cpufreq драйвер), так и прочих компонент системы (компоненты devfreq драйвера).

3. Создана модель, определяющая влияние частот процессора и прочих компонент системы на производительность процессора для заданной рабочей нагрузки.

4. Разработано решение в подсистеме планировщика ядра Linux, использующее предлагаемую модель для контроля частоты процессорных ядер.

Содержание

1 Введение

Для улучшения производительности компьютерных систем, а также для уменьшения потребляемой этими системами энергии (т.е. затрачиваемой мощности), в различных операционных системах используют механизмы регулирования тактовых частот как центрального процессора (ЦП), так и прочих компонент (графический процессор, нейронный процессор, и т.д.). Такие механизмы оперируют с технологией DVFS (Dynamic Voltage-Frequency Scaling), которая позволяет изменять тактовую частоту электронной схемы (одновременно изменяя рабочее напряжения схемы).

В операционной системе Linux (версии 6.1), представляющей собой монолитное ядро, частоты ядер процессора, как правило, меняются посредством использования драйвера *cpufreq*, предоставляющего API (Application Programming Interface) для регистрации интерфейса регулирования тактовых частот ядер ЦП. За частоты прочих компонент системы отвечает драйвер-API *devfreq* (например, регулирование частоты шины, соединяющей соседние уровни кешей, или регулирование частоты работы оперативной памяти).

На данный момент Linux использует политики регулирования тактовых частот ядер ЦП, которые не учитывают сложную структуру взаимодействия ЦП с прочими компонентами компьютерной системы, работающими при тактовой частоте, отличной от ЦП: кеши высоких уровней (L3 кеш, системный кеш), оперативная память. В современных реалиях производительность ядер ЦП в основном ограничена производительностью компонент памяти (кеши, оперативная память), взаимодействующих с ЦП, поэтому текущие политики регулирования тактовых частот в Linux не применимы в общем случае.

Отсутствие учёта компонент памяти при регулировании тактовых частот ядер ЦП является критичным при выполнении приложений реального времени, такие как компьютерные игры и мессенджеры, где выставление неподходящих тактовых частот приводит к заметному замедлению приложения (деградации производительности) и/или к повышенному энергопотреблению, что негативно сказывается на опыт пользователя.

Для учёта влияния внепроцессорных компонент на производительность ЦП необходимо построение модели производительности ядер ЦП, которое вызывает трудностей:

1. Необходимо априорное знание структуры компьютерной системы, на которой запущена операционная система, например, количество уровней кешей, их внутреннее устройство.

2. Необходимо знать подробные характеристики блоков ядра ЦП, их внутреннее устройство, для учёта их доли влияния на конечную производительность самого ядра в целом.

3. Необходимо измерить характеристики внепроцессорных компонент системы, например, зависимость задержки памяти (латентность) от уровня её утилизации (уровень нагрузки её пропускной способности), для учёта их доли влияния на конечную производительность ядра ЦП.

4. Такая информация, как утилизация оперативной памяти, или какие блоки ядра процессора были использованы за фиксированный интервал времени, получить во время работы операционной системы, как правило, невозможно, поэтому требуется обходить использование такой информации в модели.

Архитектура ARM big.LITTLE представляет собой концепцию, в которой одновременно используются различные типы ядер процессора (герогенная архитектура): ЦП

разделяется на кластеры, часть которых состоит из более энергоэффективных, но менее производительных ядер, а другая часть состоит из производительных ядер, потребляющих значительно большее количество энергии.

В компьютерных системах, использующих архитектуру ARM big.LITTLE, которая доминирует на рынке мобильных платформ на сегодняшний день, параллельно с задачей регулирования тактовых частот процессорных ядер возникает задача выбора ядерного кластера в зависимости от требований и специфики выполняемой задачи, что так же требует создание модели производительности ядер ЦП, делая эту задачу наиболее актуальной.

Для решения обозначенных задач удобно использовать симулятор компьютерных архитектур с открытым исходным кодом, например, Gem5, который и будет использоваться в данной работе. Gem5 эмулирует поведение компьютерной системы, подобной реальной системе, включая поведение ядер процессора, поведение оперативной памяти, и т.д. Среди большого числа поддерживаемых архитектур в данной работы будет использована ARM64 (к которой относится ARM big.LITTLE).

Для возможности построения модели производительности ядер ЦП, а также для реализации политики регулирования частот в операционной системе Linux, запущенной в эмуляции Gem5, были реализованы:

1. *cpufreq* драйвер под платформу эмулятора Gem5.

2. Сбор статистики микроархитектурных событий в Gem5 (универсальных для любой архитектуры), относящихся к кешам, для возможности использования этой статистики в Linux через драйвер *perf*.

Дополнительно для общности реализован драйвер-интерфейс $devfreq$ для возможности регулирования тактовых частот прочих компонент системы (при их поддержке технологии DVFS).

Реализации, выполненные в данной работе, откроют возможность сторонним разработчикам/исследователям собирать данные микроархитектурных событий, связанных с кешами, для различных тактовых частот ядер ЦП, позволяя им проводить собственные исследования в области производительности ЦП.

2 Постановка задачи

3 Обзор существующих решений

3.1 Регулирование тактовых частот ЦП

3.1.1 Технология DVFS

Почти все электронные схемы являются логическими элементами, которые имеют источник тактирования и работают на определённой частоте. Однако некоторым устройствам не выгодно работать при строго фиксированной частоте, например, если ядро процессора часто ждёт операции ввода-вывода, то эффективнее снизить тактовую частоту для более экономного расходования энергии, не сильно потеряв в производительности.

Технология DVFS (Dynamic Voltage-Frequency Scaling) позволяет в режиме реального времени регулировать тактовую частоту и напряжение, с которыми работает устройство. Как правило, возможные значения тактовых частот являются дискретным набором, причём для каждой тактовой частоты существует минимальное значение напряжения, при котором устройство всё ещё остаётся в работоспособном состоянии.

Практически все современные процессоры используют DVFS для регулировки тактовых частот ядер. В случае гетерогенных систем, т.е. имеющих кластеры ядер с различными характеристиками (например, более энергоэффективные или более производительные ядра), коими на сегодняшний день являются большинство мобильных компьютерных систем, предоставляется возможным регулировать тактовые частоты только целиком всего кластера, то есть всех ядерв кластере одновременно, а не каждого ядра по отдельности.

Регулирование тактовых частот и напряжений используется не только в случае процессорных ядер, но также и в других компонентах компьютерных систем: ОЗУ, шины, кеши и т.д..

Наибольший интерес в данной работе представляет алгоритм регулирования тактовых частот процессорных ядер, регулирование тактовых частот прочих компонент рассматриваться не будет.

3.1.2 Политики регулирования частот в ядре Linux

Единственным способом регулирования тактовых частот ядер процессора в ядре Linux является использование драйвера $cpufreq$, который предоставляет несколько политик регулирования [1]:

1. ”Performance” – тактовая частота ядра процессора выставляется в максимальное значение.

2. ”Powersave” – тактовая частота ядра процессора выставляется в минимальное значение.

3. ”Userspace” – тактовая частота ядра процессора выставляется пользователем.

4. ”Ondemand” – тактовая частота ядра процессора выставляется в зависимости от доли времени использования ядра за фиксированный промежуток времени (далее обозначается как утилизации ядерного времени).

5. ”Conservative” – поведение аналогично политике ”Ondemand”, но частота меняется менее резко, небольшими шагами.

6. "Schedutil" [2] – поведение аналогично политике "Ondemand", но утилизация отслеживается не для ядер процессора, а для каждого потока исполнения; используется более продвинутая система подсчёта утилизации ядерного времени для потока исполнения: не только за фиксированный промежуток времени, а за несколько таких промежутков подряд с учётом весов каждого промежутка времени (более давние промежутки менее важны); в случае использования не CFS (Completely Fair Scheduler) планировщика, который является стандартным в Linux версии 6.1, а дедлайн планировщика (DL) или реального времени (RT), тактовая частота выставляется в максимальное значение.

Политика "Performance" приводит к излишнему энергопотреблению, "Powersave" – к потере производительности, "Ondemand" и "Conservative" – не отслеживают утилизацию ядерного времени отдельно по потокам исполнения, к тому же не учитывают, что производительность ядра ЦП (которая обратно пропорциональна утилизации ядерного времени) может быть нелинейна относительно тактовой частоты (например, если процессор проводит основное время в ожидании выполнения транзакции-чтения из памяти), поэтому могут выставлять заведомо завышенное значения тактовой частоты, что приводит к дополнительному энергопотреблению, или медленно реагировать на повышение утилизации ядерного времени, что приводит к деградации производительности.

"Schedutil" устраняет большинство недостатков политик регулирования частот, рассмотренных выше, но всё же он не учитывает возможную нелинейность производительности ЦП от тактовой частоты ядра процессора.

Ещё один существенный недостаток всех политик, рассмотренных выше, является предположение, что частота любого ядра процессора может изменяться независимо от остальных. Как было упомянуто в 3.1.1, в современных мобильных системах ядра группируются в кластеры и частоты могут меняться только целиком для всего кластера.

3.1.3 Подсчёт утилизации ядерного времени в Linux

В ядре Linux вкрсии 6.1 в стандартном планировщике CFS (Completely Fair Scheduler) используется решение [3], в котором при подсчёте утилизации ядерного времени приложением учитывается тот факт, что ядра различного типа (в разных кластерах) имеют разную производительность, и что в рамках одного ядра ЦП производительность меняется в зависимости от выбранной частоты.

Однако авторы такого решения полностью пренебрегли тем фактом, что отношение максимальных производительностей двух различных ядер не является константой, а сильно зависит от исполняемой рабочей нагрузки (приложения). Также не учтено, что зависимость производительности, измеряемой в количестве исполненных инструкций в единицу времени, обычно нелинейна относительно тактовой частоты ядра процессора (так как скорость исполнения инструкций зависит не только от тактовой частоты ядра, но и характеристик компонент памяти).

Таким образом, авторы вводят величину ёмкости производительности ядра процессора, измеряемой количеством исполненных инструкций в единицу времени при заданной тактовой частоте при условии максимальной утилизации ядерного времени. Величина утилизации вводится как часть ёмкости производительности, использованной при заданной утилизации ядерного времени (т.е. меньше утилизация ядерного времени – меньше величина утилизации).

Более формально, если за фиксированный интервал времени τ ядро процессора было задействовано время Δt, то утилизация ядерного времени равна $\Delta t/\tau$, а для i-ого ядра процессора формула утилизации имеет следующий вид:

$$util = \frac{\Delta t}{\tau} \cdot \frac{freq_{cpu_i}}{freq_{cpu_i}^{max}} \cdot \frac{capacity_{cpu_i}}{\max_i\{capacity_{cpu_i}\}} = inv_{i,freq}, \quad (1)$$

где $freq_{cpu_i}$ – тактовая частота ядра процессора, при которой была измерена утилизация ядерного времени Δt, $freq_{cpu_i}^{max}$ – максимально возможная тактовая частота ядра процессора, $capacity_{cpu_i}$ – ёмкость производительности i-ого ядра.

Ёмкости производительности вычисляются единожды с помощью измерений для конкретных сценариев исполнения (приложений) и применяются во всех остальных случаях без каких-либо обоснований, причём с линейной зависимостью производительности от тактовой частоты ядра, что неверно в большистве случаев.

В реальности используется утилизация, подсчитанная не за один промежутков времени, а сразу за несколько промежутков времени (обычно одинаковых по длительности), например, PELT (Per Entity Load Tracking) [2] использует экспоненциально движующееся среднее значение утилизаций, подсчитанных по формулам выше.

=———————— далее проверить

В качестве альтернативы PELT, который официально представлен в ядре Linux, компанией Google был разработан WALT (Window Assisted Load Tracking) [4], который предназначен специально для мобильных устройств, где важна быстрая реакция со стороны ядра на изменение нагрузки процессора, чтобы не потерять в производительности. WALT считает утилизацию за существенно меньшее количество промежутков времени (5 вместо 32) и без экспоненциально движущихся средних значений, но алгоритм подсчёта утилизации за 1 промежуток времени совпадает с описанным выше.

При выборе политики регулирования частот в Linux "Schedutil" [2], WALT или PELT используются не только внутри планировщика, но и для выбора частоты ядра процессора, т.е. "Schedutil" переиспользует подсчитанное значение утилизации потока исполнения для регулирования частот (только в случае CFS планировщика).

Таким образом, в данное время ни стандартный планировщик, ни подсистемы регулирования частот ядра Linux не учитывают особенности влияния рабочих нагрузок на производительность ядер процессора.

3.1.4 Альтернативные алгоритмы регулирования частот

[тут очередной обзор литературы и пример работ на схожую тематику]

3.2 Симулятор архитектуры Gem5

При исследовании компьютерных архитектур, их оптимизации или оценки новых идей, связанных с параметрами таких архитектур, чаще всего используют не конечное устройство, а симуляцию архитектуры такого устройства, т.к. это и дешевле в разработке, и открывает больше возможностей в плане вариации параметров компонент архитектуры при оценке их влияния на разрабатываемое решение (оптимизация, алгоритм и т.д.).

Одним из наиболее популярных и продвинутых симуляторов компьютерных систем является Gem5 [5]. Он поддерживает различные архитектуры: ARM, ALPHA, MIPS, Power, SPARC, x86 и т.д.. Для симуляции возможно использовать несколько типов процессоров [6], среди которых "Atomic Simple", "Timing Simple", "Minor", "O3". "Atomic Simple" является функциональной симуляцией машинных инструкций, "Timing Simple" – простейшей потактовой модели исполнения машинных инструкций, "Minor" – модель процессора, не поддерживающего спекулятивное исполнение (Out-of-Order), а "O3" – поддерживающего спекулятивное исполнение.

Gem5 поддерживает 2 режима симуляции: эмуляция системных вызовов (SE – syscall emulation), который поддерживает симуляцию одного приложения с некоторыми ограничениями (например, ввиду отсутствия таблицы страниц, поддерживаемой операционной системой, обращения в виртуальную память упрощено, MMU (Memory Management Unit) не участвует в симуляции), и полная эмуляция системы (FS – full system).

В данной работе используется архитектура ARM, т.к. именно она используется в большинстве мобильных систем на сегодняшний день. В качестве процессора выбран процессор "Minor", т.к. он обеспечивает хорошую точность симуляции и на практике процессоры такого типа реально используются в мобильных устройствах в качестве энергоэффективных ядер. Процессор "O3" обладает более сложным конвейером и поддерживает внеочередное исполнение инструкций, такие ядра обычно используют как производительные (с повышенным энгергопотреблением), но симуляция такого типа процессорного ядра занимает больше времени, чем "Minor", поэтому он не выбран чисто из практических соображений.

Компанией ARM в симуляторе Gem5 было разработано процессорное ядро HPI (High Performance In-order) [7] на основе "Minor" специально для исследовательских целей. В дальнейшем будет использован именно этот тип процессорного ядра.

В качестве режима симуляции выбрана полная эмуляция системы, которая поддерживает полноценный запуск ядра Linux под архитектуру ARM. Именно в режиме полной эмуляции будет произведено дальнейшее исследование-построение модели для регулирования частот процессорных ядер.

4 Исследование и построение решения задачи

4.1 Описание модели производительности процессора

Главной характеристикой производительности процессора является количество инструкций, исполняемых в еденицу времени – чем больше это значение, тем быстрее исполняется рабочая нагрузка (программа). Причём заданное количество инструкций исполняется за разное количество процессорных циклов, которые, в свою очередь, обратно пропорциональны времени исполнения этих инструкций.

Пусть за время τ процессор непрерывно исполнял инструкции и исполнил $instrs$ инструкций за $cycles$ циклов при заданной частоте процессора $freq_{cpu}$. Тогда, очевидно, выполняется следующее соотношение:

$$cycles = \frac{freq_{cpu}}{\tau} \tag{2}$$

На практике чаще всего используют такие величины как $cpi = cycles/instrs$ и $ipc = instrs/cycles$ в качестве меры производительности процессора: чем выше/ниже значение ipc/cpi, тем лучше процессор. Однако кроме типа процессорного ядра на эти значения также влияют частота самого ядра и частоты остальных компонент системы. Например, при повышении частоты процессора величина ipc либо остаётся такой же, если отсутствуют инструкции, связанные с обращением высокие уровни памяти (ОЗУ или кеши высокого уровня), либо уменьшается, если процессор часто обращается в кеши высокого уровня или оперативную память, из-за чего циклы тратятся впустую.

За время τ процессор часть циклов тратит на исполнение инструкций непосредственно на процессорных блоках (в том числе вычислительных), а оставшуюся часть на ожидание операций, связанных с обращением в память (в кеши или оперативную память): обозначим эти величины $cycles_{cpu}$ и $cycles_{mem}$, тогда справедливо

$$cpi = \frac{cycles}{instrs} = \frac{cycles_{cpu}}{instrs} + \frac{cycles_{mem}}{instrs} \tag{3}$$

Важно отметить, что $\frac{cycles_{cpu}}{instrs}$ – значение cpi, если бы все обращения в кеши и оперативную память занимали 0 циклов, т.е. эти работали бы бесконечно быстро, а значит количество циклов ограничивалось снизу возможностями процессора. Обозначим данное соотношение как $cpi_{cpu} \equiv \frac{cycles_{cpu}}{instrs}$.

В свою очередь величина $cycles_{mem}$ характеризуется только лишь внешними компонентами системы, не зависящими от конвейера ядра процессора. Пусть, например, если имеется 2 уровня кешей и оперативная память. Время доступа к определённому уровню кеша характеризуется средней величиной $cycles_{L_i}$, представляющую собой латентность кеша (время задержки обращения), выраженную в процессорных циклах, где i – номер уровня кеша. После обращения в кеш возможны 2 ситуации: либо попадание к кеш, либо промах и обращение в следующий уровень кеша или оперативную память. Время доступа к оперативной памяти обозначим $cycles_{ram}$.

Выражая времена доступа через собственные частоты и латентности, получим:

$$cycles_{mem} = n_{L_1} \cdot \frac{lat_{L_1}}{freq_{L_1}} \cdot freq_{cpu} + n_{L_2} \cdot \frac{lat_{L_2}}{freq_{L_2}} \cdot freq_{cpu} + n_{ram} \cdot \frac{lat_{ram}}{freq_{ram}} \cdot freq_{cpu}, \tag{4}$$

$$cpi = cpi_{cpu} + \frac{n_{L_1} \cdot \frac{lat_{L_1}}{freq_{L_1}} + n_{L_2} \cdot \frac{lat_{L_2}}{freq_{L_2}} + n_{ram} \cdot \frac{lat_{ram}}{freq_{ram}}}{instrs} \cdot freq_{cpu}, \tag{5}$$

где lat_{L_1}, lat_{L_2} и lat_{ram} – латентности, выраженные в собственных циклах (а не процессорных). n_{L_1}, n_{L_2} и n_{ram} – характерные количества обращений к соответствующим уровням памяти в случае промаха в предыдущий уровень памяти и попадания в текущий за время τ. Заметим, что эти значения не являются абсолютными значениями количества обращений, так как при обращении в память присутствует параллелизм, как было отмечано в **??**. Значит эти значения вкладывают как параллелизм обращений, так и их количество.

Заметим, что из формулы следует, что повышение частоты процессора ведёт к увеличению *cpi* и уменьшению *ipc*, а повышение частот кешей и оперативной памяти, наоборот, ведёт к уменьшению *cpi* и увеличению *ipc*.

Чаще всего уровни кешей, наиболее близкие к процессорному ядру, имеют такой же источник тактирования, что и процессорное ядро, т.е. такие кеши оперируют на тех же частотах, что и сам процессор. Например, если бы кеш первого уровня оперировал с частотами процессора ($freq_{L_1} = freq_{cpu}$), тогда формулу можно упростить до более простой:

$$cpi = cpi_{cpu} + n_{L_1} \cdot \frac{lat_{L_1}}{instrs} + \frac{n_{L_2} \cdot \frac{lat_{L_2}}{freq_{L_2}} + n_{ram} \cdot \frac{lat_{ram}}{freq_{ram}}}{instrs} \cdot freq_{cpu} = \tag{6}$$

$$= cpi_{cpu}^{L_1} + \frac{n_{L_2} \cdot \frac{lat_{L_2}}{freq_{L_2}} + n_{ram} \cdot \frac{lat_{ram}}{freq_{ram}}}{instrs} \cdot freq_{cpu}, \tag{7}$$

то есть можно занести константную часть для заданной нагрузки, независимую от каких-либо частот, в cpi_{cpu}, что упрощает формулу. Однако, теперь $cpi_{cpu}^{L_1}$ зависит не только от характера утилизации нагрузкой процессорных блоков, как cpi_{cpu}, но также и от характеристик кеша первого уровня и количества обращений в него (т.е. от уровня его утилизации).

Латентности кешей и оперативной памяти могут и не быть константными значениями, т.к. могут зависеть от характера и частоты обращений к ним, как отмечано в **??**. Так, латентность оперативной памяти определяется как текущим уровнем утилизации этой памяти, так и особенностями обращения к ней (например, в случае DDR памяти, где элементарными ячейками являются банки данных, состоящие из строк и столбцов, существует кеш строки, который значительно влияет на скорость обращения к ячейке памяти, т.е. локальность обращения к ячейке DDR памяти очень важна).

У описываемой модели есть ряд преимуществ, которые делают её лучше аналогичных моделей:

1. Построив модели, с помощью которых можно вычислять lat_{L_2} и lat_{ram}, в режиме реального времени можно вычислить величину $cpi_{cpu}^{L_1}$ для заданной нагрузки без построения соответствующей модели: достаточно знать значения $cycles$, $instrs$, n_{L_2} и n_{ram}.

2. Зная величину $cpi_{cpu}^{L_1}$, можно вычислить *cpi* (а значит и *ipc*) для любого набора частот процессора и прочих компонент, частоты которых входят в вышеописываемую формулу, причём не требуются знания о характере событий, происходящих внутри процессора, и даже о событиях, происходящих внутри кеша первого уровня.

3. В случае физического ядра, разделённого на 2 виртуальных (так называемый SMT - Simultaneous multithreading), по-прежнему можно пользоваться формулой выше, т.к. становится неважно, как именно виртуальные ядра разделяют между собой

процессорные блоки и как именно они взаимодействуют с кешом первого уровня – эти аспекты учитываются в формуле и не требуют дополнительных вычислений.

Таким образом, для применения модели необходимо:

1. Иметь готовые модели для поиска lat_{L_2}, lat_{ram}.
2. Уметь каким-либо образом вычислять значения $cycles$, $instrs$, n_{L_2} и n_{ram}.
3. Знать списки возможных частот $freq_{cpu}$, $freq_{ram}$, $freq_{L_2}$.

4.2 Модель производительности применительно к архитектуре ARM

[перепроверить всё, что идёт дальше]

Рассмотрим способ применения описанной выше модели в случае архитектуры ARM. Для подсчёта значений $cycles$, $instrs$, n_{L_2} и n_{ram} можно ипользовать следующие PMU счётчики процессора:

1. CPU_CYCLES – количество затраченных процессорных циклов;
2. $INST\_RETIRED$ – количество исполненных инструкций;
3. $L1I\_CACHE\_REFILL$ – количество чтений инструкций (instruction fetches), которые отсутствуют в кеше инструкций уровня L1 (промах в L1 кеш инструкций), поэтому инструкции вынуждены читаться из кешей более высокого уровня. Некешируемые промахи в кеш и операции синхронизации кешей не считаются;
4. $L1D\_CACHE\_REFILL$ – количество чтений и записей данных (data loads and stores), или операций обращений в таблицу страниц (page table walks), которые не смогли найти нужную информацию в кеше данных уровня L1 (промах в L1 кеш данных), поэтому данные вынуждены выгружаться из кешей более высокого уровня. Некешируемые промахи в кеш и операции синхронизации кешей не считаются;
5. $L2D\_CACHE\_REFILL$ – работает как сумма счётиков $L1D\_CACHE\_REFILL$ и $L1I\_CACHE\_REFILL$, но применительно к уровню кеша L2. В случае промаха в кеш дальнейшее обращение может происходить не в кеш более высокого уровня, если таковой отсутствует, а сразу в оперативную память.

В счётчиках типа $\_CACHE\_REFILL$ существует очень важный нюанс, не оговорённый в спецификациях PMU счётчиков компании ARM Ltd., но упоминаемый в иных документах: в платформах архитектуры ARM не существует PMU счётчиков, которые считают промахи в кеши, вместо них используются счётчики, считающие количество перезапонений кеш-линий (кешами более выского уровня ил оперативной памятью). Возможна ситуация, когда один промах в кеш вызывает несколько перезаполнений кеш-линий (например, если промах случился по адресу, который пересекает 2 соседние кеш-линии), или наоборот: несколько промахов в кеш могут быть разрешены благодаря одному перезаполнению кеш-линии.

Заметим, что чаще всего в современных мобильных системах на архитектуре ARM после последнего уровня процессорного кеша располагают дополнительный кеш – системный кеш (system cache), который предназначен для периферийных устройств. Он может быть как больше по размеру, чем последний уровень процессорного кеша, так

и меньше. Наличия системного кеша ведёт к тому, что для определения значения n_{ram} недостаточно воспользоваться счётчиками, описанными выше.

Единственный счётчик, который умеет считать количество промахов в кеш, является $LL\_CACHE\_MISS\_RD$. Причиной тому является то, что кеш последнего уровня, как правило, является системным кешом, который находится за пределами CPU на чипе. Данный счётчик считает количество промахов, когда в результате данные берутся из оперативной памяти, из другого чипа или из соседнего процессорного кластера с использованием технологии Intercluster peering. Нас интересует только первый случай.

Согласно документации ARM, если счётчик $LL\_CACHE\_MISS\_RD$ не реализован, то есть бит $EXTLLC$ в регистре $CPUECTLR$ выставлен в ноль, то значения $LL\_CACHE\_MISS\_RD$ будут совпадать со значениями $L2D\_CACHE\_REFILL$, так что в случае наличия системного кеша такая ситуация может заметно ухудшить работу модели (по-прежнему предполагается система из 2 уровней кеша и возможно системного кеша).

4.3 Модель производительности применительно к ядру Linux

4.4 Реализация модели в планировщике ядра Linux

5 Описание практической части

6 Заключение

Список литературы

[1] Documentation Linux Kernel. — Linux CPUfreq governors : 2008. — https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt.

[2] Documentation Linux Kernel. — Schedutil : 2013. — https://docs.kernel.org/scheduler/schedutil.html.

[3] Documentation Linux Kernel. — Capacity Aware Scheduling : 2024. — https://docs.kernel.org/scheduler/sched-capacity.html.

[4] Qualcomm Technologies Inc. — WALT vs PELT : Redux : 2017. — https://static.linaro.org/connect/sfo17/Presentations/SFO17-307%20WALT%20vs%20PELT.pdf.

[5] Binkert Nathan, Beckmann Bradford, Black Gabriel, Reinhardt Steven K, Saidi Ali, Basu Arkaprava, Hestness Joel, Hower Derek R, Krishna Tushar, Sardashti Somayeh, et al. The gem5 simulator // ACM SIGARCH computer architecture news. — 2011. — Vol. 39, no. 2. — P. 1–7.

[6] Andreas Sandberg, Stephan Diestelhorst, and William Wang. — Architectural Exploration with gem5 : 2017. — https://www.gem5.org/assets/files/ASPLOS2017_gem5_tutorial.pdf.

[7] Research Ashkan Tousi | Arm. — System Modeling using gem5 : 2017. — https://old.gem5.org/wiki/images/c/cf/Summit2017_starterkit.pdf.

[8] Ltd. Arm. — Arm Neoverse N1 Core: Performance Analysis Methodology : 2021.

[9] Overview Cortex A77. — Cortex A77 Documentation : 2020. — https://en.wikichip.org/wiki/arm_holdings/microarchitectures/cortex-a77.

[10] Clapp Russell, Dimitrov Martin, Kumar Karthik, Viswanathan Vish, and Willhalm Thomas. Quantifying the performance impact of memory latency and bandwidth for big data workloads // 2015 IEEE International Symposium on Workload Characterization / IEEE. — 2015. — P. 213–224.

[11] Keramidas Georgios, Spiliopoulos Vasileios, and Kaxiras Stefanos. Interval-based models for run-time DVFS orchestration in superscalar processors // Proceedings of the 7th ACM international conference on Computing frontiers. — 2010. — P. 287–296.

[12] Eyerman Stijn, Heirman Wim, and Hur Ibrahim. DRAM bandwidth and latency stacks: Visualizing DRAM bottlenecks // 2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) / IEEE. — 2022. — P. 322–331.

[13] David Howard, Fallin Chris, Gorbatov Eugene, Hanebutte Ulf R, and Mutlu Onur. Memory power management via dynamic voltage/frequency scaling // Proceedings of the 8th ACM international conference on Autonomic computing. — 2011. — P. 31–40.

Приложение

Список используемых обозначений и терминов

1. $cycles$ – количество циклов процессорного ядра за определённый промежуток времени;
2. $instrs$ – количество исполненных инструкций процессорным ядром за определённый промежуток времени;
3. $ipc = \frac{instrs}{cycles}$, $cpi = \frac{cycles}{instrs}$;
4. OoO (Out-of-Order) – парадигма в современных высокопроизводительных CPU, основанная на спекулятивном исполнении инструкций для повышения значения ipc;
5. ROB (Re-order buffer) – циклический буфер, используемый для работы алгоритма Томасуло для поддержки OoO в CPU;
6. SoC (System-on-Chip) – интегральная схема, включающая в себя большинство или все компоненты компьютерной системы, такие как CPU, интерфейсы памяти, устройства ввода-вывода и так далее;
7. DDR SDRAM (Double Data Rate Synchronous Dynamic Random-Access Memory) – динамическая оперативная память синхронного доступа с двойной скоростью передачи данных – класс интегральных схем памяти, используемой в компьютерах (в качестве ОЗУ);
8. LPDDR SDRAM (Low-Power Double Data Rate Synchronous Dynamic Random-Access Memory) – динамическая оперативная память синхронного доступа с двойной скоростью передачи данных и с низким энергопотреблением – класс интегральных схем памяти, используемой в мобильных компьютерах (в качестве ОЗУ);
9. DDR (Double Data Rate) – общее обозначение, которое подразумевает DDR SDRAM или LPDDR SDRAM память;
10. DVFS (Dynamic Voltage-Frequency Scaling) – технология, позволяющая регулировать рабочие частоту и напряжение устройства в режиме реального времени.