

Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 21.Б10-мм

Оптимизация библиотеки XXHASH для архитектуры RISC-V

Пономарев Николай Алексеевич

Отчёт по учебной практике
в форме «Эксперимент»

Научный руководитель:
ст. преподаватель кафедры ИАС К. К. Смирнов

Санкт-Петербург
2023

Оглавление

Введение	3
1. Постановка задачи	4
2. Обзор	5
2.1. Векторное расширение RISC-V	5
3. Реализация	6
4. Эксперимент	10
Заключение	11
Список литературы	12

Введение

xxHASH¹ — современная библиотека для хеширования, целью которой является генерация хеша со скоростью, сравнимой со скоростью оперативной памяти. RISC-V, в свою очередь, это молодая и активно развивающаяся архитектура, которая получила поддержку от таких компаний как GOOGLE [1], IMAGINATION TECHNOLOGIES [4] и ALIBABA [3].

Высокую скорость работы, в частности, для хешей XXH3 и XXH128, обеспечивает реализация алгоритмов хеширования с помощью векторных расширений процессора. Некоторые процессоры архитектуры RISC-V имеют поддержку векторных инструкций. К сожалению, долгое время спецификация векторного расширения находилась в разработке, что привело к наличию в продаже процессоров с разными версиями расширения [2].

Библиотека xxHASH ещё не содержит в себе оптимизаций с помощью векторного расширения для платформы RISC-V (сокращенно RVV). На момент написания данной работы в продаже можно было найти только процессоры с «урезанной» поддержкой RVV. Изучение вопроса о том, возможно ли ускорение алгоритмов при переносе на процессоры с «ограниченными» возможностями на примере библиотеки xxHASH, является целью данной работы.

В работе приводится изучение различий в версиях RVV, разбор трудностей, встреченных при переносе на процессор с «ограниченными» возможностями, и исследование результатов измерения производительности.

¹<https://github.com/Cyan4973/xxHash>

1. Постановка задачи

Целью данной работы является оптимизация библиотеки XXHASH при помощи векторных расширений архитектуры RISC-V.²

Для достижения данной цели были поставлены следующие задачи:

- Сравнить возможности разных версий векторного расширения RISC-V;
- Выбрать целевую платформу для адаптации кода и проведения измерений;
- Адаптировать одну из существующих реализаций под выбранную платформу;
- Выполнить замеры производительности оптимизированного кода;
- Исследовать причины замедления.³

²TODO: если меряем «в минус», то цель примерно такая «исследование возможности оптимизации библиотеки XXHASH с помощью „урезанного“ RVV»

³TODO: опять если «в минус», и в целом это можно только предполагать, ибо а) некогда; б) как мерить циклы проца?

2. Обзор

2.1. Векторное расширение RISC-V

Векторное расширение RISC-V находилось в разработке довольно долго, из-за чего производители чипов начали выпускать свои решения основываясь на недоработанной версии стандарта 0.7.1. Один из таких чипов — ALLWINNER D1, получил большое распространение в недорогих одноплатных ПК, таких как Sipeed Lichee RV и MangoPi MQ-Pro.

Последней и официальной ратифицированной версией является версия 1.0.

3. Реализация

Библиотека содержит в себе четыре алгоритма хеширования: XXH32, XXH64, XXH3, XXH128. Первые два из них не поддаются оптимизации с помощью векторных операций, поэтому интерес представляют только последние. Внутри они используют функции `XXH3_accumulate` и `XXH3_scrambleAcc`. Именно эти функции могут использовать векторные возможности процессора: в библиотеке уже имеется поддержка SSE2, AVX512, NEON и других. Сама библиотека написана на языке C, а для оптимизации применяются intrinsic функции.

Уже существующие реализации оперируют компонентами вектора размером в 64 бита, в терминологии RISC-V это называется Selected Element Width (SEW). А длина вектора разнится от 128 бит до 512 бит, в терминологии RISC-V — Vector Length (VL).

По спецификации векторного расширения RISC-V, минимальный VL равен 128 битам, также должна присутствовать поддержка SEW равного 64 битам. К сожалению, на момент написания данной работы в продаже можно было найти лишь устройства на чипе ALLWINNER D1, в котором отсутствует поддержка 64-битных элементов вектора, поэтому была предпринята попытка использовать SEW в 32 бита из-за чего потребовалась некоторое количество ухищрений.

В качестве эталонной реализации был выбран вариант для набора команд SSE2, т. к. целевой процессор имеет VL в 128 бит, как и SSE2.

Одной из первых проблем стала операция умножения. В алгоритме требуется перемножить 2 вектора, используя только младшие 32 бита каждого элемента, в результате чего получаются 64-битные числа. При SEW в 32 бита необходимо перемножить между собой нулевой и второй элементы каждого вектора. Для совершения данной операции пришлось использовать отдельные инструкции для вычисления младших и старших битов произведения, затем сдвигать старшие биты в первый и третий элемент вектора, а после объединять их по маске, таким образом, чтобы в нулевой и второй элемент попали младшие биты произведения, а в первый и третий старшие соответственно. Данную

Листинг 1: Умножение с ручным расширением до 64 бит

```
vuint32m1_t product_hi = vmulhu(data_key, data_key_lo, VL);  
↪ // Вычисление старших битов произведения  
vuint32m1_t product_lo = vmul(data_key, data_key_lo, VL);  
↪ // Вычисление младших битов произведения  
vuint32m1_t product_hi_sl = vrgather(product_hi,  
↪ xlshift_mask, VL); // Сдвиг старших битов  
vuint32m1_t product = vmerge(xmerge_mask, product_hi_sl,  
↪ product_lo, VL); // Объединение двух векторов в один
```

операцию иллюстрирует листинг 1.

Кроме того, сложение тоже требует модификации: необходимо сохранить бит переноса, т. к. мы складываем 64-битные числа, представленные как пары 32-битных. RISC-V имеет инструкцию, которая по паре векторов заполняет битовую маску: если при сложении необходим перенос, соответствующий бит становится единицей. Однако команды для сдвига маски не существует, поэтому используется такая последовательность команд: сначала вычисляется маска переноса, затем, с помощью операции И в маске оставляются только биты на нулевом и втором местах, после этого выполняется обычное сложение. Затем чтобы прибавить бит переноса, элементы вектора попарно переставляются местами, производится прибавление переноса, а затем возвращение элементов на свои места. Эту операцию иллюстрирует листинг 2.

ALLWINNER D1 поддерживает векторное расширение RISC-V экспериментальной версии 0.7.1, которая более не имеет официальной поддержки, а единственный компилятор со встроенной поддержкой поддерживается компанией ALIBABA на основе GCC 10. Актуальная и ратифицированная версия расширения — 1.0, она поддерживается современными версиями LLVM и GCC. К сожалению, в версии 0.7.1 гораздо менее доработана, нежели версия 1.0.

Одной из проблем версии 0.7.1, является отсутствие операции загрузки маски из памяти. Для обхода этого ограничения используется

Листинг 2: Сложение с переносом

```
vbool32_t carry_bits = vmadc(prod_even, prod_odd,  
    ↪ xzero_mask, VL); // Вычисляем маску битов переноса  
carry_bits = vmand(carry_bits, xmerge_mask, VL);  
    ↪ // Оставляем в маске только биты на 0 и 2 местах  
vuint32m1_t prod = vadd(prod_even, prod_odd, VL);  
    ↪ // Складываем числа без переноса  
prod = vrgather(prod, xswap_mask32, VL);  
    ↪ // Меняем попарно местами элементы вектора  
prod = vmadc(prod, xzero_vector, carry_bits, VL);  
    ↪ // Прибавляем биты переноса  
prod = vrgather(prod, xswap_mask32, VL);  
    ↪ // Возвращаем элементы вектора на свои места
```

команда `vmseq.vx`, она принимает на вход вектор и число, и если элемент вектора равен заданному числу, то бит в маске устанавливается в единицу.

Операция вычисления битов переноса при сложении в версии 0.7.1 сделана таким образом, что ей всегда требуется маска, даже если это первое вычисление и поэтому необходимо создавать маску из нулей для корректной работы.

Современные версии LLVM и GCC поддерживают «перегрузку» интринсик функций, что делает код гораздо более читаем — строго типы нужно указывать малому количеству операции, например загрузке и выгрузке векторов в память. К сожалению, в компиляторе от ALIBABA данная возможность отсутствует.

Спецификация векторных расширений RISC-V не определяет корректное поведение при загрузке данных из памяти по невыровненному адресу. Так например в функции `XXH3_accumulate`, два из трех указателей, принимаемых на вход, являются невыровненными. К счастью, эту проблему легко обойти: достаточно загружать данные, как если бы вектор был из 8-битных элементов, а затем интерпретировать его как вектор нужной размерности.

Листинг 3: Сравнение перегруженных и не перегруженных функций

```
// Код без использования "перегрузки"
vbool32_t carry_bits = vmadc_vvm_u32m1_b32(prod_even,
↪ prod_odd, xzero_mask, VL);
// Код с использованием "перегрузки"
vbool32_t carry_bits = vmadc(prod_even, prod_odd,
↪ xzero_mask, VL);
```

Листинг 4: Загрузка по невыровненному адресу

```
vuint32m1_t key_vec =
↪ vreinterpret_v_u8m1_u32m1(vle8_v_u8m1((uint8_t*)(xsecret
↪ + VL * i), VL * 4));
```

4. Эксперимент

Измерения проводились на одноплатном компьютере Sipeed Lichee RV 86 со следующими характеристиками:

- Процессор ALLWINNER D1 с частотой 1 ГГц;
- Оперативная память DDR3 объемом 512 Мб с частотой 792 МГц;
- Операционная система DEBIAN SID с последними обновлениями на момент тестирования.

Для компиляции всегда использовались флаги `-O3 TODO`, а для выбора набора функций использовались флаги `-DXXH_VECTOR=XXH_SCALAR` и `-DXXH_VECTOR=XXH_RVV` соответственно.

TODO: Таблички с замерами

TODO: А что ж всё так плохо-то?

Заключение

В рамках данной учебной практики были достигнуты следующие результаты:

- TBD: зависит от постановки задачи

Исходный код расположен по адресу: <https://github.com/WoWaster/xxHash>.

Список литературы

- [1] Google Announces Official Android Support for RISC-V | Ars Technica. — URL: <https://arstechnica.com/gadgets/2023/01/google-announces-official-android-support-for-risc-v/> (дата обращения: 2023-05-14).
- [2] Implications of Widely Distributed Draft 0.7.1 Implementation(s) · Issue #667 · Riscv/Riscv-v-Spec, GitHub. — URL: <https://github.com/riscv/riscv-v-spec/issues/667> (дата обращения: 2023-05-14).
- [3] Robinson Dan. Alibaba Launches RISC-V Developer Platform for Edge SoCs. — URL: https://www.theregister.com/2022/08/25/alibaba_riscv_developer_platform/ (дата обращения: 2023-05-14).
- [4] Why We've Levelled up on RISC-V. — URL: <https://blog.imaginationtech.com/why-weve-levelled-up-on-risc-v> (дата обращения: 2023-05-14).