Оптимизация библиотеки xxHash для архитектуры RISC-V

Пономарев Николай

RISC-V — молодая и активно развивающаяся архитектура. Для экосистемы любой архитектуры очень важно наличие широкоиспользуемых библиотек и оптимизации для них. xxHash — современная библиотека для хеширования, скорость которой обеспечивается использование векторных инструкций процессора. Так, например, в библиотеке уже имеется поддержка наборов инструкий SSE2, AVX512, NEON.

Однако поддержки векторного расширения RISC-V — RVV — в библиотеке пока что нет, что и стало целью работы.

Разработка процессоров — сложное занятие, и выпущенные процессоры обычно отстают от актуальных спецификаций архитектуры, поэтому существует две несовместимые между собой версии RVV, доступных исследователям:

- RVV 1.0 — официально принятая версия. Тем не менее на данный момент нет ни одной аппаратной платформы, доступной для покупки, где она была бы реализована;
- RVV 0.7.1 — бета-версия расширения, которую можно найти в процессорах от T-Head.

В моём распоряжении была плата Lichee Pi 4A с поддержкой RVV 0.7.1, на которой и проводились эксперименты.

Основная идея работы была простой: смотря на уже существующие оптимизации, написать код для RVV для обеих версий RVV с использованием intrinsic функций. Однако было встречено несколько трудностей:

- RVV 0.7.1 поддерживает только форк компилятора GCC от компании T-Head, а современные версии GCC и Clang поддерживают только RVV 1.0;
- Новые версии компиляторов (GCC $\geqslant$ 13, Clang $\geqslant$ 16) используют префикс `__riscv` для intrinsic функций, что при написании кода для обеих версий стандарта требует магии с макросами;

- В RVV 0.7.1 отсутствуют некоторые инструкции, присутствующие в RVV 1.0, что увеличивает объем кода.

После реализации оптимизированных функций появилась необходимость измерить производительность написанного кода. Здесь стоит сказать о том, что для выбора набора оптимизаций используются возможности макропроцессора, а также что библиотека имеет встроенный бенчмарк, позволяющий легко сравнивать различные реализации. Сначала была произведена попытка измерить производительность с помощью QEMU, но она не увенчалась успехом в силу того, что QEMU производит все векторные операции, используя скалярные регистры. После этого было проведено тестирование на плате. С его результатами можно ознакомиться в пулл-реквесте.

Сам пулл-реквест можно найти по ссылке: `https://github.com/Cyan4973/xxHash/pull/898`.