Как ускорить Eigen на RISC-V?

Зайцева Ксения,
Младший инженер-программист
компании YADRO

Алгоритмы Eigen

Операции над плотными объектами

- Базовые манипуляции;
- Арифметические операции;
- Поиск минимума, сумма/произведение элементов;
- Выделение подматрицы, репликации, reverse;
- Покомпонентные математические STL-like операции (тригонометрия, log, power).

Линейная алгебра и разложения

- Разложения: FullPivLU, PartialPivLU, HouseholderQR (ColPiv, FullPiv), UTV, LLT, LDLT, BDCSVD, JacobiSVD;
- Решение системы;
- Поиск собственных чисел и векторов;
- Ранг матрицы;
- Обратная матрица;
- Определитель.

Разреженные матрицы

- Базовые манипуляции;
- Арифметические операции (sparse-dense и sparse-sparse);
- Выделение подматрицы;
- Покомпонентные операции;
- Солвера: LLT, LDLT, LU, QR, BiCGSTAB, ConjugateGradient, LeastSquares ConjugateGradient.

Геометрический модуль

- Абстрактные преобразования: повороты, сдвиги, масштабирование;
- Матрицы проективных или аффинных преобразований;
- Векторное произведение;
- Ортогональный вектор.

Выделенные алгоритмы содержат низкоуровневые оптимизации.

Реализация Eigen

- Header-only библиотека основанная на шаблонах выражений;
- Использует «ленивые вычисления»;
- Иерархия классов основана на Curiously Recurring Template Pattern (CRTP);

- Низкоуровневые оптимизации также организованы в концепции шаблонов.
- Содержит оптимизации под SSE, AVX, AVX2, AVX512, AltiVec/VSX (в системах Power7/8), ARM NEON, ARM SVE, S390x SIMD (ZVector), MIPS SIMD (MSA).

Для интринсиков различных расширений используются общие обертки:

SIMD 128-bits

```
template<> EIGEN_STRONG_INLINE
    Packet4f pmul<Packet4f>(const Packet4f& a,
                            const Packet4f& b) {
      return _mm_mul_ps(a,b);
}
```

SSE

```
template<> EIGEN_STRONG_INLINE
    Packet4f pmul<Packet4f>(const Packet4f& a,
                            const Packet4f& b) {
      return vmulq_f32(a,b);
}
```

NEON

```
template <> EIGEN_STRONG_INLINE
  PacketXf pmul<PacketXf>(const PacketXf& a,
                          const PacketXf& b) {
    return svmul_f32_z(svptrue_b32(), a, b);
}
```

SVE

SSE, AVX, SVE

RVV

Performance

Summary

SSE

Работают с пакетами по 128 бит - 4 int, 4 float или 2 double.

Переопределяемые обертки:

- mul/div/add/sub/fma;
- <, >, ==, !=;
- &, |, xor, <<, >>, max, min;
- load, store;
- reductions: max, min, add, mul;
- scatter, gather;
- transpose, reverse;
- abs, floor, casting

и тд.

```
typedef __m128 Packet4f;
typedef __m128d Packet2d;
typedef __m128i Packet4i;
```

```
Packet2cf ~ Packet4f
Packet1cd ~ Packet2d
```

```
template<> EIGEN_STRONG_INLINE
    Packet4f pmul<Packet4f>(const Packet4f& a,
                            const Packet4f& b) {
      return _mm_mul_ps(a,b);
}
```

AVX

Работает с пакетами по 256 бит - 8 int, 8 float или 4 double.

Поскольку AVX можно работать как с пакетами по 32 байта так и по 16 байт, для матриц фиксированного размера оптимальный размер пакета определяется автоматически.

Для векторов и матриц с динамическими размерами будут использоваться только 32-байтовые пакеты. Вектора меньшего размера будут по возможности векторизованы с помощью SSE.

```
typedef __m256 Packet8f;
typedef __m256d Packet4d;
typedef __m256i Packet8i;
```

```
Packet4cf ~ Packet8f
Packet2cd ~ Packet4d
```

```
template <> EIGEN_STRONG_INLINE
  Packet8f pmul<Packet8f>(const Packet8f& a,
                          const Packet8f& b) {
    return _mm256_mul_ps(a, b);
}
```

Arm's Scalable Vector Extension (SVE)

Поскольку SVE довольно молод, пользователь должен включить его явно.

Работает с пакетами по 128-1024 бит.

Ширина векторного регистра задается во время компиляции: -msve-vector-bits=N

Пока содержит только пакеты PacketXf и PacketXi. Их размер зависит от длины вектора SVE. Например, если установлено -msve-vector-bits=512, PacketXf будет содержать 512/32 = 16 элементов.

```
typedef svfloat32_t PacketXf;
typedef svint32_t PacketXi;
```

```
template <> EIGEN_STRONG_INLINE
  PacketXf pmul<PacketXf>(const PacketXf& a,
                          const PacketXf& b) {
    return svmul_f32_z(svptrue_b32(), a, b);
}
```

SSE, AVX, SVE

RVV

Performance

Summary

Атрибут riscv_rvv_vector_bits

Этот атрибут используется для определения типов фиксированной длины для одного из существующих безразмерных типов RVV.

```
typedef vfloat32m2_t fixed_vfloat32m2_t __attribute__((riscv_rvv_vector_bits(256)));
```

Значение аргумента должно быть равно ширине векторного регистра в битах на LMUL (aka 2(m2) * 128 = 256 для vfloat32m2).

Ширина векторного регистра задается опцией компиляции -mrvv-vector-bits=128 или парой -mrvv-vector-bits=zvl, -march=rv64gcv_zvl128b + предопределяется макрос __riscv_v_fixed_vlen

Для fixed_vfloat*m*_t разрешены операции:

- sizeof
- Глобальная переменная
- Элемент класса, структуры
- Приведение к другим эквивалентным типам
- CMP: >, <, ==, !=, <=, >=
- ALU: +, -, *, /, -

Eigen RVV

- Пока содержит только пакеты PacketXf и PacketXi;
- Только LMUL=1;
- Включается явно;
- Работает с пакетами по 128-1024 бит;
- RVV1.0 + RVV0.7.1
- Ширина векторного регистра задается во время компиляции: -mrvv-vector-bits=N или -mrvv-vector-bits=zvl, -march=rv64gcv_zvl128b

```
template <> EIGEN_STRONG_INLINE
  PacketXf pmul<PacketXf>(const PacketXf& a,
                          const PacketXf& b) {
    return __riscv_vfmul_vv_f32m1(a, b, packet_traits<float>::size);
}
```

```
typedef vfloat32m1_t PacketXf __attribute__((riscv_rvv_vector_bits(__riscv_v_fixed_vlen)));

typedef vint32m1_t PacketXi __attribute__((riscv_rvv_vector_bits(__riscv_v_fixed_vlen)));
```

SSE, AVX, SVE

RVV

Performance

Summary

Как проводились замеры?

- Прирост от векторизации в Eigen сравнивался с приростом библиотеки OpenBLAS, которая также содержит низкоуровневые оптимизации под RISC-V

- Замеры проводились для каждой библиотеки в 3-х случаях:
  - скалярный код (отключена автовекторизация)
  - с автовекторизацией
  - сборка с имплементированным векторным кодом

- Компилятор - LLVM Syntacore DevToolkit 2024.06. Более ранние версии (до обновления на llvm-18) содержат баг, приводящий к серьезной деградации производительности.

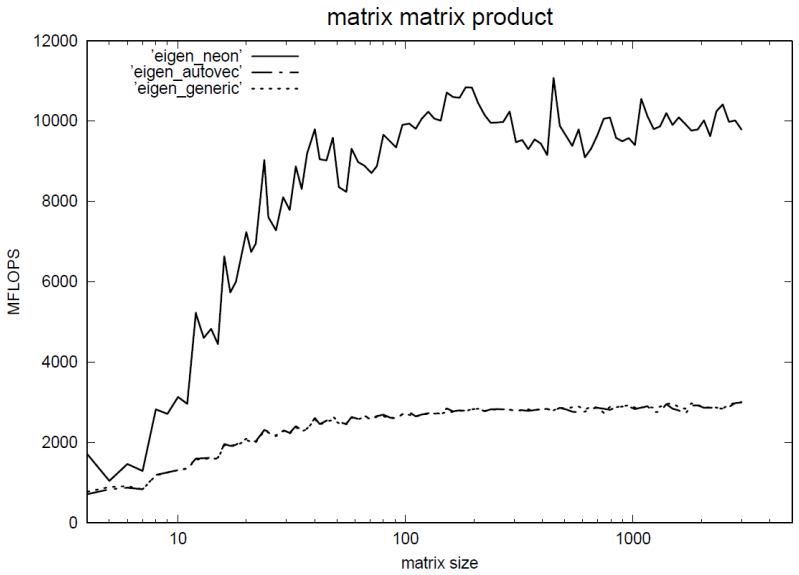
	Cortex-A73	Kendryte K230	BananaPI	LicheePi 4A
Архитектура	ARM64	RISCV64	RISCV64	RISCV64
Векторное расширение	NEON 128b	RVV1.0 128b	RVV1.0 256b	RVV0.7.1 128b
Тактовая частота	2.2Gz	1.6Gz	1.6Gz	1.85Gz
RAM	8GB	512MB	4GB	16GB
Компилятор	GNU gcc-11.4	Syntacore toolchain 24.06	Syntacore toolchain 24.06	T-head toolchain v2.8.1

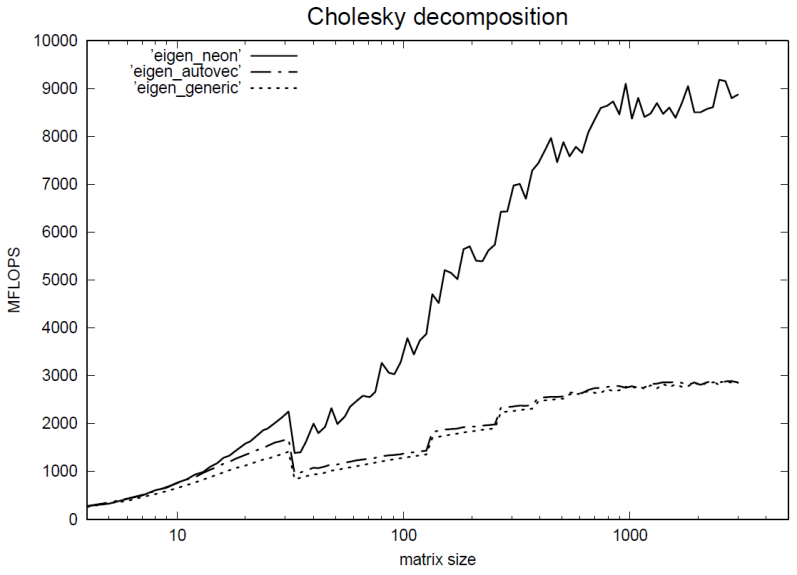
Cortex-A73

- NEON, 128 bits
- Просадка относительно автовекторизованного кода на BLAS level1
- На BLAS level2-3 LAPACK автовекторизации почти не происходит

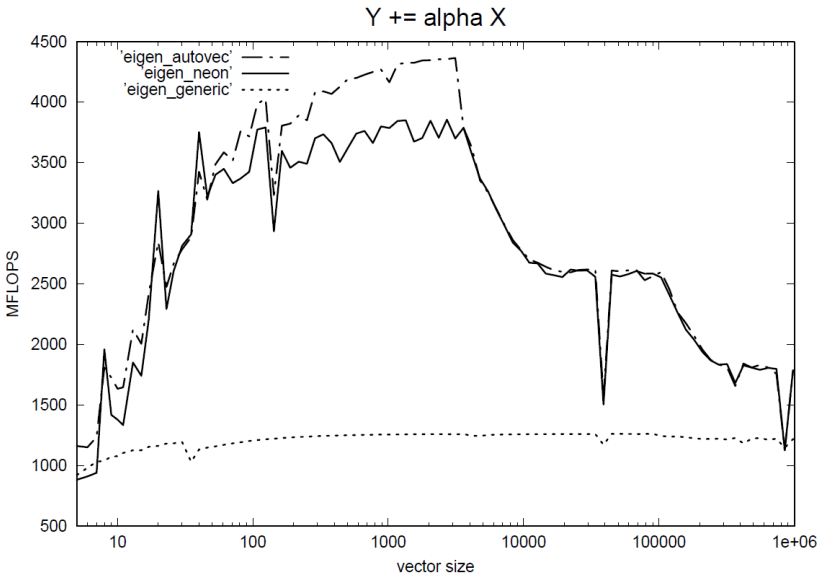
Cortex A73	BLAS level1		LAPACK, BLAS level2-3	
	to scalar	to autovec	to scalar	to autovec
Eigen	x2.0	x0.9	x2.3	x2.2

*В среднем по алгоритмам и размерностям





BLAS Level2-3, LAPACK



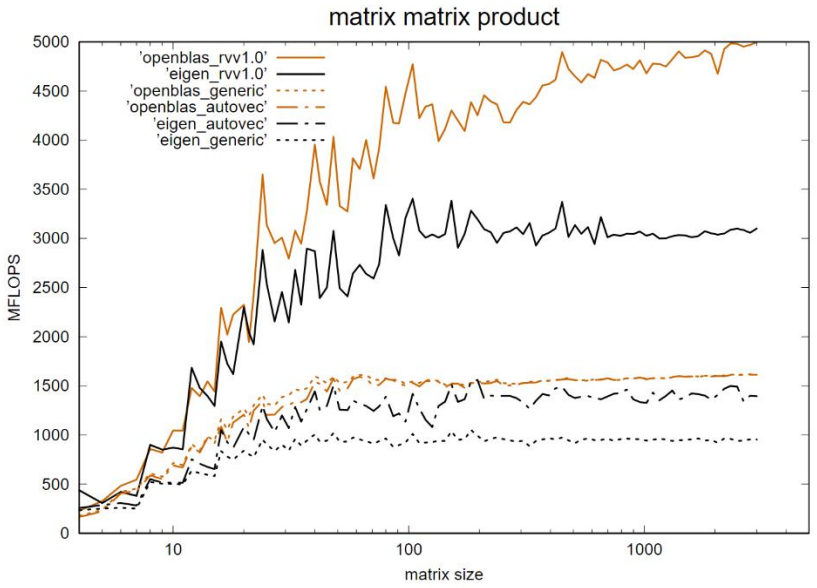
BLAS Level1

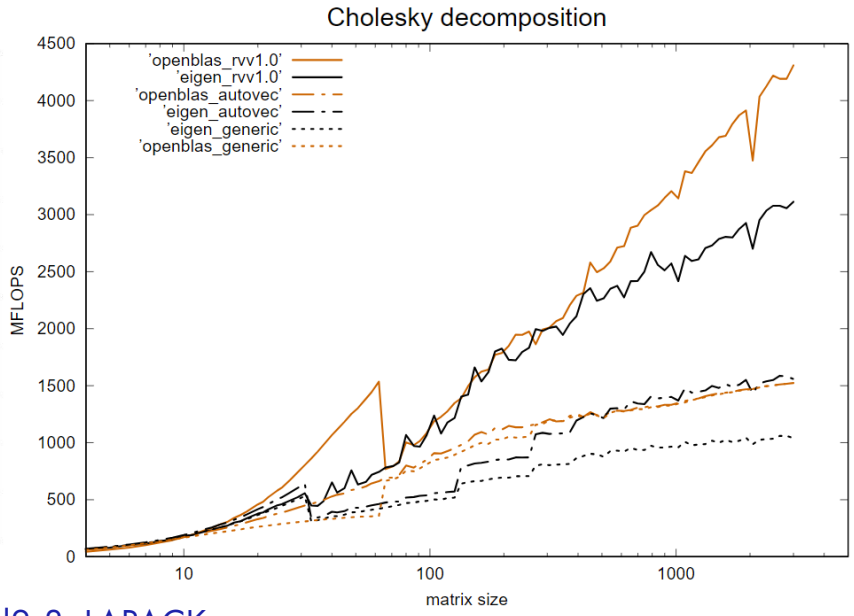
Kendryte K230

- RVV1.0, VLEN=128
- Просадка относительно автовекторизованного кода на BLAS level1
- На BLAS level2-3, LAPACK прирост меньше чем на NEON примерно на 10%

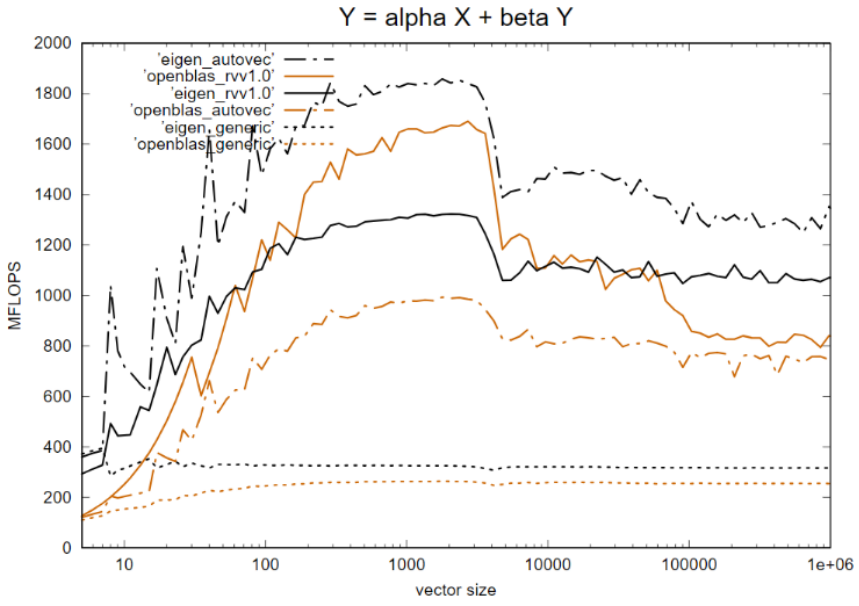
Kendryte K230	BLAS level1		LAPACK, BLAS level2-3	
	to scalar	to autovec	to scalar	to autovec
Eigen	x2.3	x0.8	x2.1	x1.9
OpenBLAS	x3.1	x1.2	x2.2	x1.9

*В среднем по алгоритмам и размерностям





BLAS Level2-3, LAPACK



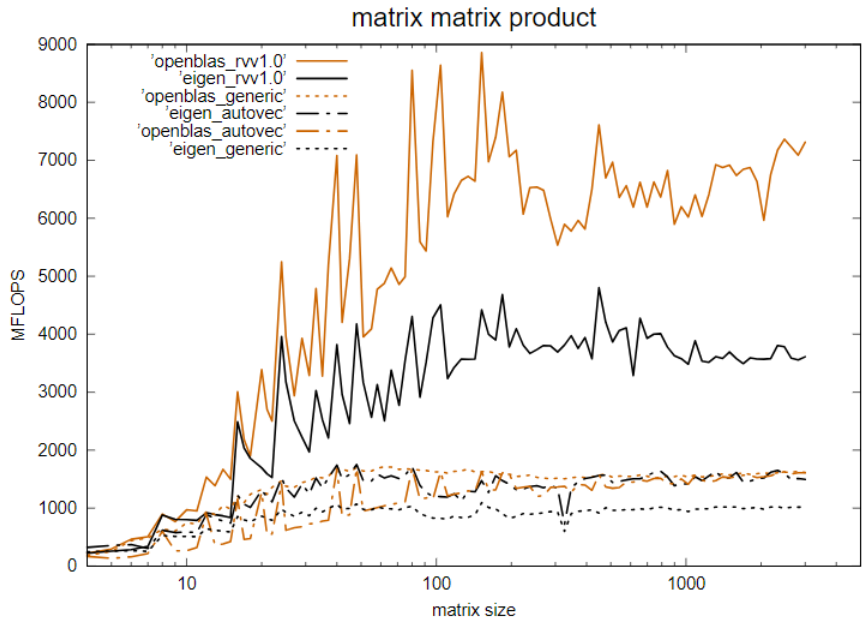
BLAS Level1

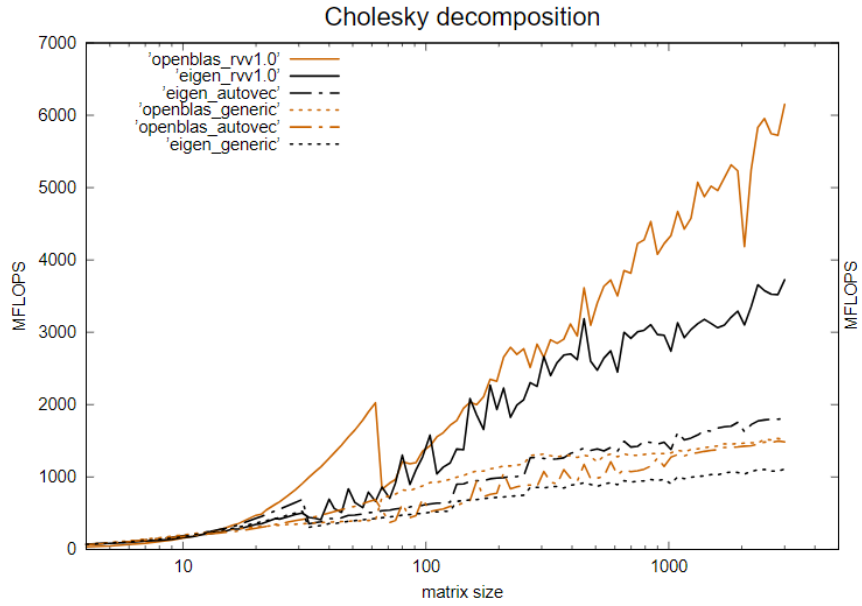
BananaPI

- RVV1.0, VLEN=256
- Сохраняется увеличение коэффициента ускорения при масштабировании ширины векторного регистра

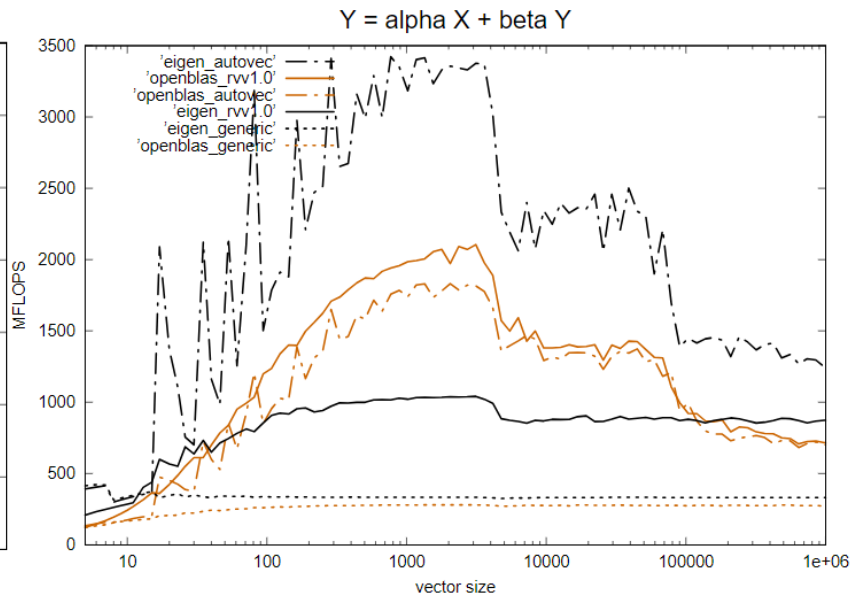
BananaPi	BLAS level1		LAPACK, BLAS level2-3	
	to scalar	to autovec	to scalar	to autovec
Eigen	x2.7	x0.8	x2.2	x1.8
OpenBLAS	x3.3	x1.1	x2.6	x2.6

*В среднем по алгоритмам и размерностям





BLAS Level2-3, LAPACK



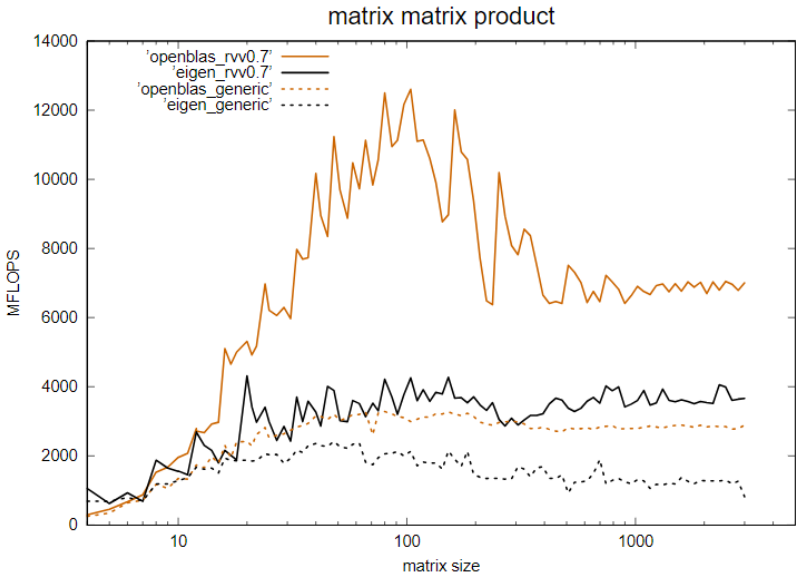
BLAS Level1

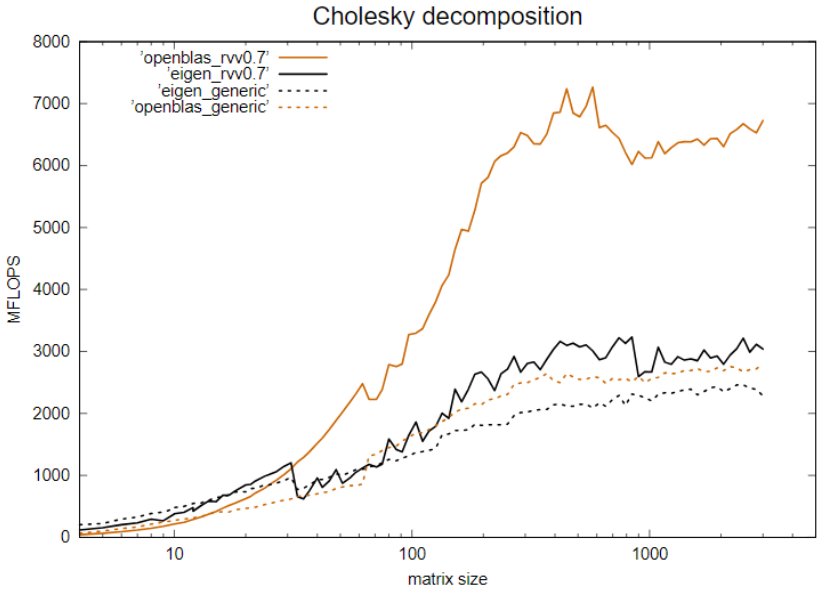
LicheePI RVV0.7.1

- VLEN=128
- Автовекторизации нет
- Флаги оптимизации -fschedule-insns (в -O2) и -fpeel-loops (в -O3) приводят к деградации производительности и/или функционально неверному исполнению. Отключены для сборки eigen_rvv0.7 на LAPACK, BLAS level2-3

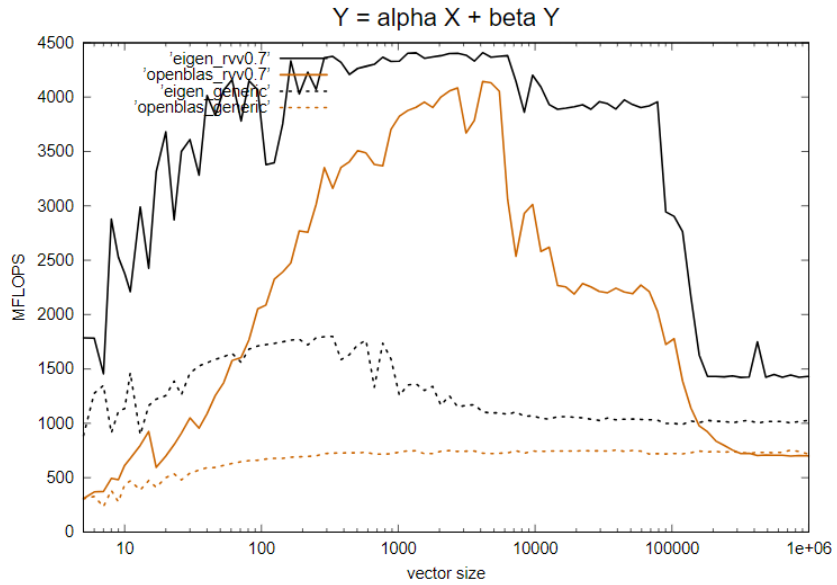
LicheePi	BLAS level1	LAPACK, BLAS level2-3
Eigen	x2	x1.5
OpenBLAS	x1.9	x1.9

*В среднем по алгоритмам и размерностям





BLAS Level2-3, LAPACK



BLAS Level1

SSE, AVX, SVE

RVV

Performance

Summary

Summary

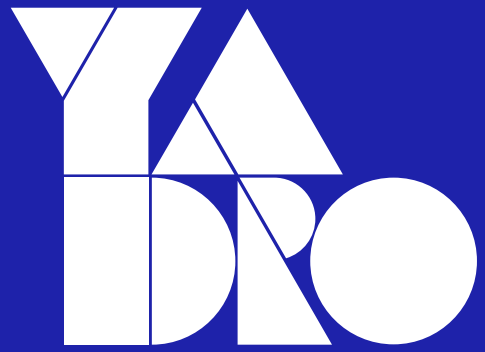
Добавлена поддержка RVV в Eigen:

- x2.2 среднее ускорение на Kendryte RVV1.0 128b
- x2.5 среднее ускорение на BananaPi RVV1.0 256b
- x1.8 среднее ускорение на LicheePI RVV0.7.1, хотя RVV0.7.1 не получил достаточно качественной поддержки fixed-vlen

Среднее ускорение от RVV1.0 в Eigen составляет ~80%, того ускорения, что в среднем дает RVV1.0 в OpenBLAS

Дальнейшие планы:

- Возможно добавление остальных типов данных: double, unsigned, long, sort, half, complex
- Выяснить причину с деградаций перфоманса относительно автовекторизации на алгоритмах BLAS level1
- Контрибьют в upstream MR-1687



Москва,
ул. Рочдельская, 15, стр. 13
+7 800 777-06-11

yadro.com