



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ

КАФЕДРА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ И СИСТЕМНОГО ПРОГРАММИРОВАНИЯ

ЛАБОРАТОРИЯ ITLav



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ
КАФЕДРА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ И СИСТЕМНОГО ПРОГРАММИРОВАНИЯ
ЛАБОРАТОРИЯ ITLAB

***ВВЕДЕНИЕ В АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ
И ОПТИМИЗАЦИЮ ПРОГРАММ***

**Оптимизация программ для
процессоров архитектуры RISC-V на
примере алгоритмов линейной алгебры**



**LOBACHEVSKY
UNIVERSITY**

приоритет2030⁺

Мееров И.Б., Пирова А.Ю.,
Волокитин В.Д., Козинов Е.А.

Содержание

□ Стандарт BLAS

- Интерфейс и обзор реализаций

□ Оптимизация функций BLAS для ленточных матриц

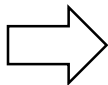
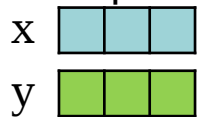
- Умножение ленточной матрицы общего вида на вектор (GBMV)
- Умножение треугольной ленточной матрицы на вектор (TBMV)
- Умножение симметричной ленточной матрицы на вектор (SBMV)
- Решение СЛАУ с треугольной ленточной матрицей (TBSV)

□ Литература

СТАНДАРТ BLAS

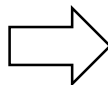
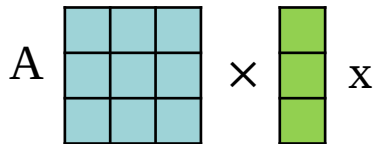
Интерфейс библиотек для базовых операций линейной алгебры (BLAS)

- ❑ **BLAS** (Basic Linear Algebra Subprograms) – интерфейс библиотек для базовых операций линейной алгебры (Lawson, Hanson et al., 1979)
- ❑ Включает функции 3 уровней, реализации для 4 типов данных
 - Уровень 1 – вектор-векторные операции (16 функций)



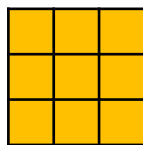
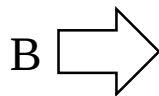
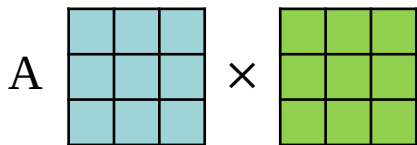
$$axpy$$
$$y \leftarrow \alpha * x + y$$

- Уровень 2 – матрично-векторные операции (25 функций)



$$gemv$$
$$y \leftarrow \alpha * op(A) * x + \beta * y$$

- Уровень 3 – матричные операции (9 функций)



$$gemm$$
$$C \leftarrow \alpha * op(A) * op(B) + \beta * C$$

Реализации стандарта BLAS

- ❑ Существует много реализаций стандарта для различных архитектур:
 - Intel oneAPI MKL, Goto BLAS, Netlib BLAS, ARMAS (процессоры x86), ACML (процессоры AMD), cuBLAS, CLBlast, ViennaCL (GPU)...
 - ATLAS, Eigen, BLIS, OpenBLAS – оптимизированы для различных архитектур (x86, IBM Power, ARM, MIPS)
 - OpenBLAS – оптимизированная реализация BLAS с открытым исходным кодом
 - Содержит низкоуровневые оптимизированные реализации функций для различных типов процессоров, в том числе, для RISC-V
 - Поддерживается большим сообществом разработчиков
- ❑ Как сравнивать разные реализации?
- ❑ Как выявить функции, реализацию которых можно улучшить?

Есть ли потенциал для оптимизации?

- ❑ **Потенциал для оптимизации есть (почти) всегда**
- ❑ Разработана **система тестирования** корректности и производительности для алгоритмов BLAS
- ❑ Выбраны тестовые данные (сгенерированные матрицы). Матрицы разбиты на «маленькие», «средние», «большие» по объему памяти для хранения:
 - «маленькие» – данные умещаются в кэш L1, «средние» – все данные не попадают в кэш L1, но при этом не выходят за кэш L3, «большие» – данных превосходят кэш L3 и полностью помещаются в оперативную память
- ❑ Проведены масштабные эксперименты на суперкомпьютере (Intel x86) для сравнения производительности OpenBLAS vs. MKL
- ❑ Обнаружено, что работа с ленточными матрицами организована в OpenBLAS **недостаточно эффективно**

Программная реализация

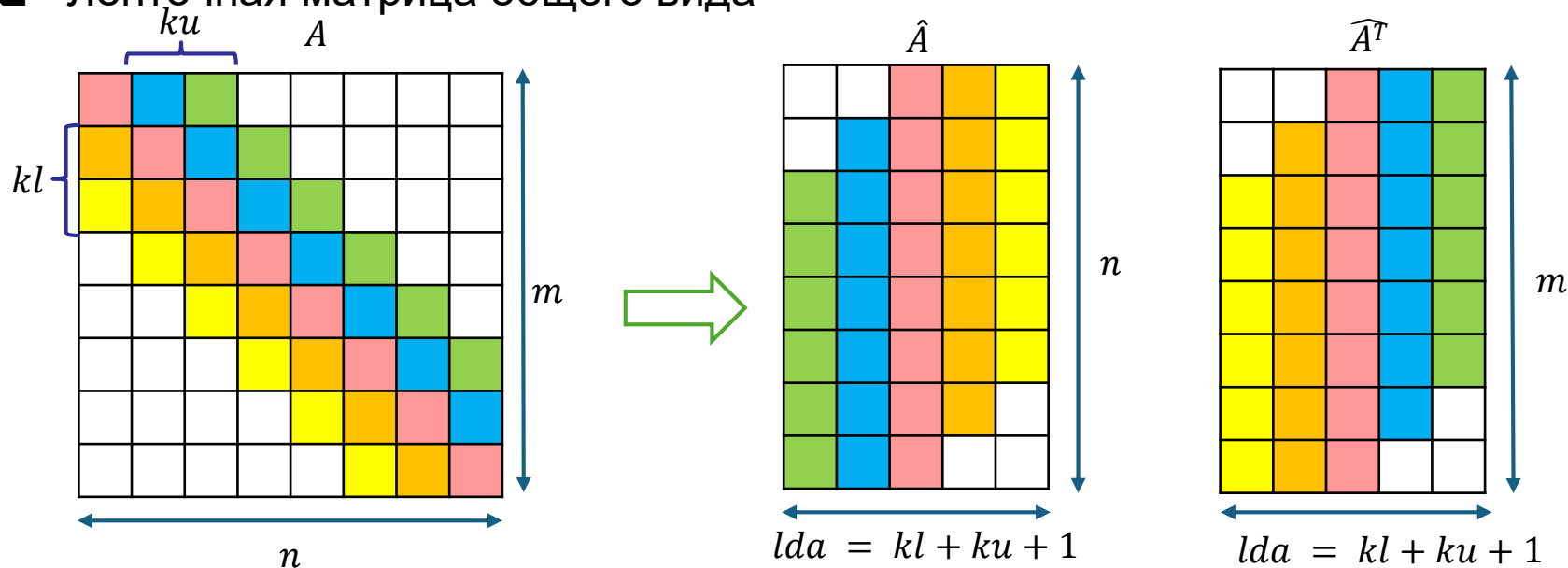
- ❑ Базовая версия – библиотека OpenBLAS
- ❑ В библиотеку OpenBLAS были встроены векторные реализации указанных функций для процессоров x86 (набор инструкций AVX-512) и RISC-V (набор инструкций rvv 0.7.1, rvv 1.0).
 - Точность одинарная и двойная
 - Последовательная версия

Функции BLAS level 2 для ленточных матриц

- ❑ **GBMV:** $y = \alpha \times op(A) \times x + \beta \times y$, где x, y – вектора, α, β – скаляры, A – ленточная матрица с kl нижними и ku верхними диагоналями размера $m \times n$, $op(A) = A$ или $op(A) = A^T$.
- ❑ **SBMV:** $y = \alpha \times A \times x + \beta \times y$, где x, y – вектора, α, β – скаляры, A – симметричная ленточная матрица с k побочными диагоналями, размера $n \times n$ (хранится только верхний или нижний треугольник).
- ❑ **TBMV:** $x = op(A) \times x$, где x, y – вектора, A – треугольная ленточная матрица с k побочными диагоналями, размера $n \times n$, $op(A) = A$ или $op(A) = A^T$.
- ❑ **TBSV:** решение СЛАУ $op(A)x = b$, где x, b – вектора, A – треугольная ленточная матрица с k побочными диагоналями размера $n \times n$, $op(A) = A$ или $op(A) = A^T$ (хранится только верхний или нижний треугольник).

Формат хранения

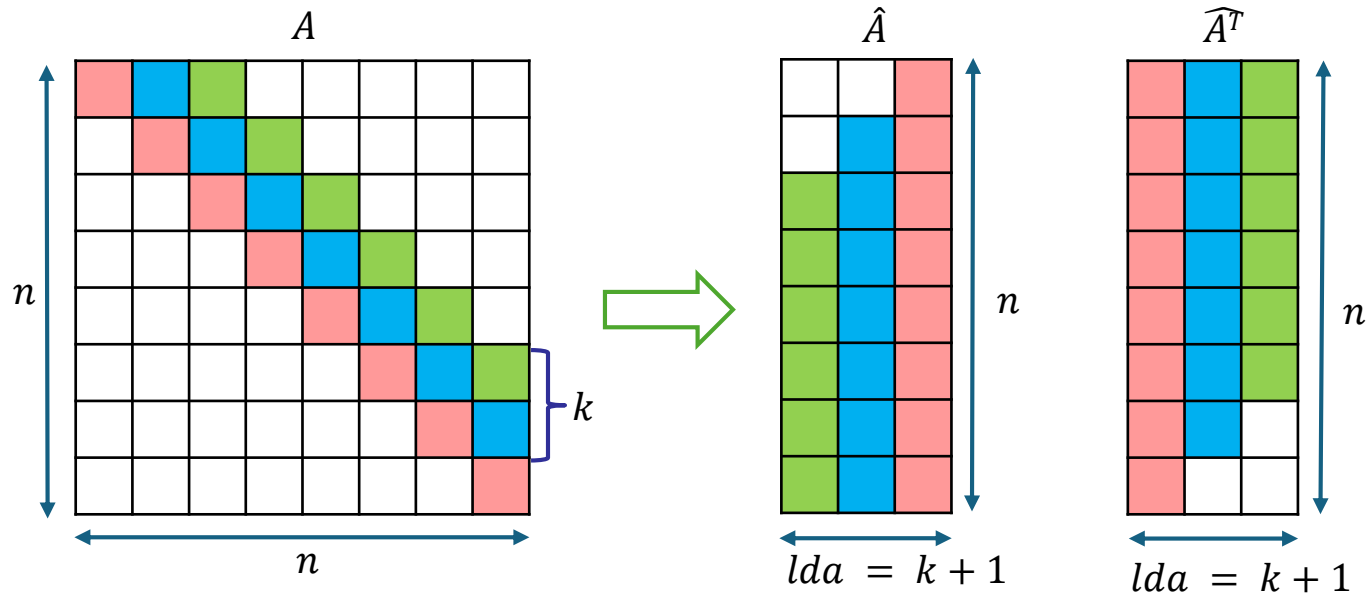
❑ Ленточная матрица общего вида



❑ Матрица хранится по столбцам

Формат хранения

- Треугольная или симметричная ленточная матрица



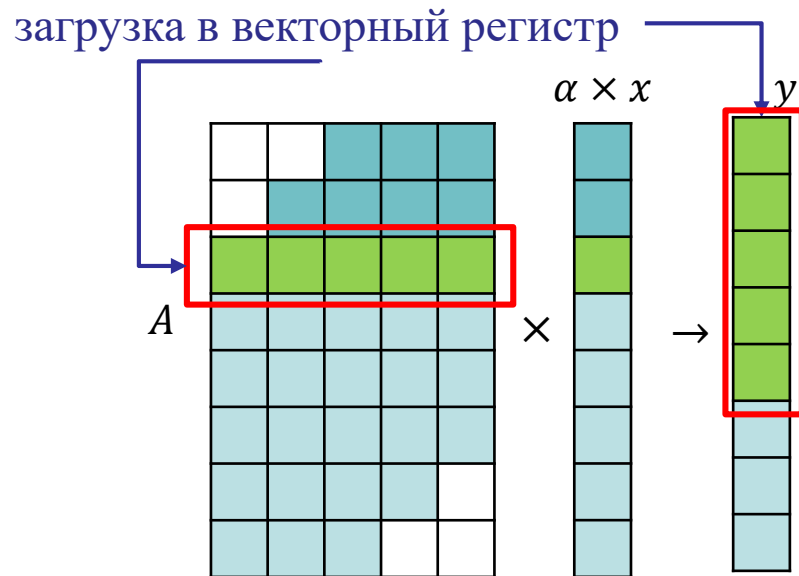


GBMV

УМНОЖЕНИЕ ЛЕНТОЧНОЙ МАТРИЦЫ ОБЩЕГО ВИДА НА ВЕКТОР

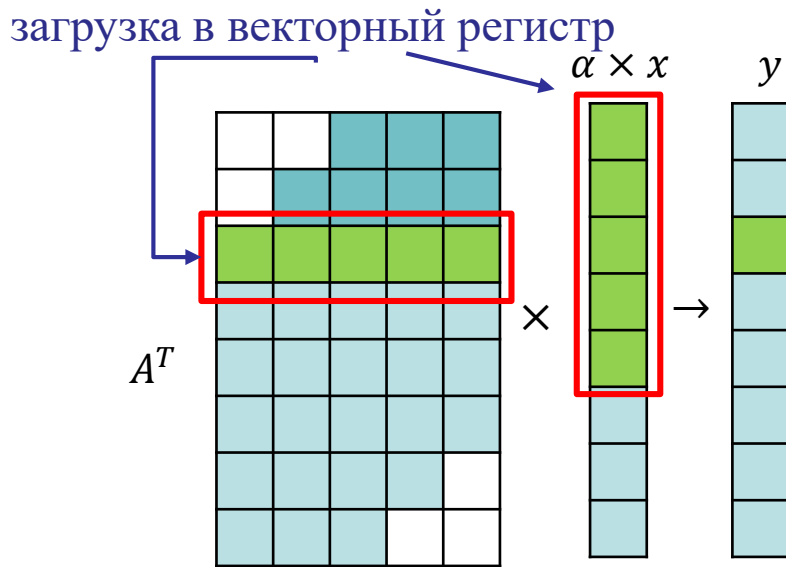
Умножение ленточной матрицы общего вида на вектор (GBMV)

- ❑ Функция GBMV: $y = \alpha \times op(A) \times x + \beta \times y$,
где x, y – вектора, α, β – скаляры, A – ленточная матрица с kl нижними и ku верхними диагоналями размера $m \times n$, $op(A) = A$ или $op(A) = A^T$.
- ❑ **Не транспонированная матрица**
- ❑ Основная вычислительная операция:
 - **AXPY**: $y = \alpha \times x + y$,
где x, y – векторы, α – скаляр
- ❑ В OpenBLAS векторизация выполнена в AXPY
- ❑ Эффективность векторизации зависит от числа диагоналей матрицы



Умножение ленточной матрицы общего вида на вектор (GBMV)

- ❑ Функция GBMV: $y = \alpha \times op(A) \times x + \beta \times y$,
где x, y – вектора, α, β – скаляры, A – ленточная матрица с kl нижними и ku верхними диагоналями размера $m \times n$, $op(A) = A$ или $op(A) = A^T$.
- ❑ Транспонированная матрица
- ❑ Основная вычислительная операция – **DOT**:
$$res = \sum_{i=1}^n x_i y_i,$$
 x, y – векторы
- ❑ В OpenBLAS векторизация выполнена в DOT



Умножение ленточной матрицы общего вида на вектор (GBMV). Базовый алгоритм

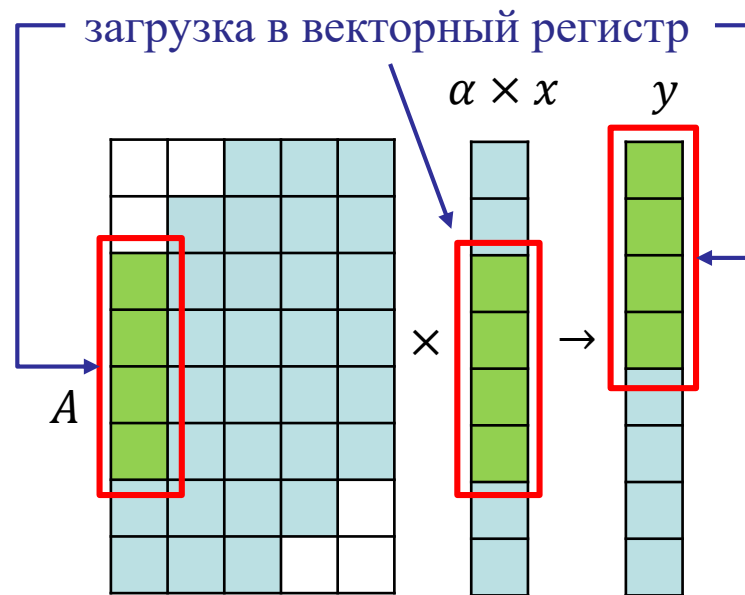
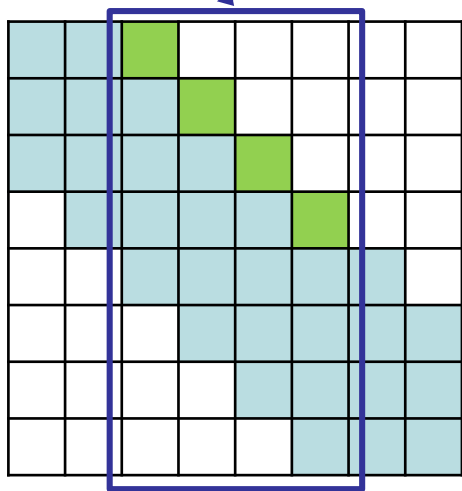
- ❑ Базовая реализация: при малом числе диагоналей вызываются скалярные версии AXPY и DOT [RoofLine](#)

```
void GBMV(BLASLONG m, BLASLONG n, BLASLONG ku, BLASLONG kl, FLOAT alpha,
          FLOAT *a, BLASLONG lda, FLOAT *X, FLOAT *Y) {
    BLASLONG offset_u = ku; BLASLONG offset_l = ku + m;
    for (i = 0; i < MIN(n, m + ku); i++) {
        start = MAX(offset_u, 0);
        end   = MIN(offset_l, ku + kl + 1);
        length = end - start;
#ifdef TRANS
        AXPYU(length, alpha * X[i], a + start, Y + start - offset_u);
#else
        Y[i] += alpha * DOT(length, a + start, X + start - offset_u);
#endif
        offset_u --; offset_l --; a += lda;
    }
}
```

Умножение ленточной матрицы общего вида на вектор (GBMV). Оптимизированный алгоритм 1...

- ❑ **Идея:** разделить матрицу на вертикальные полосы, обход выполнять по диагоналям матрицы

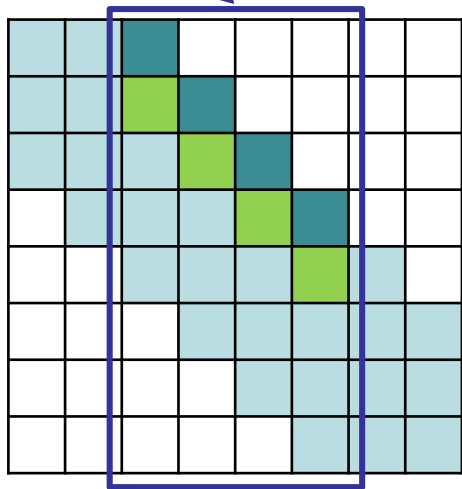
ширина полосы = длине векторного регистра



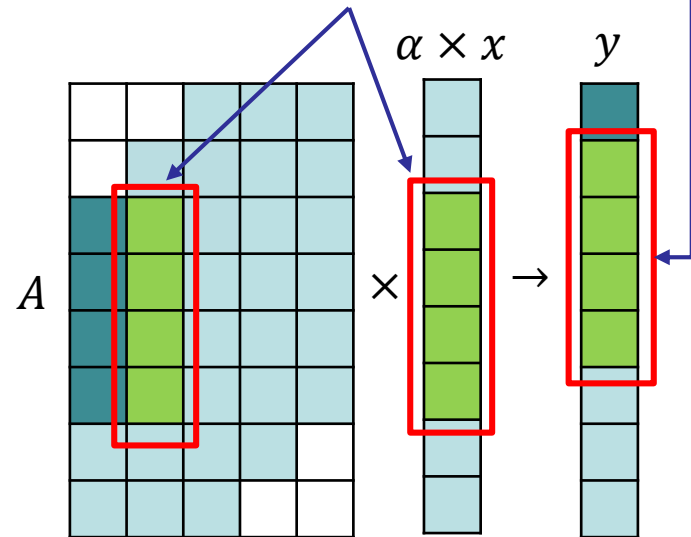
Умножение ленточной матрицы общего вида на вектор (GBMV). Оптимизированный алгоритм 1...

- ❑ **Идея:** разделить матрицу на вертикальные полосы, обход выполнять по диагоналям матрицы

ширина полосы = длине векторного регистра



загрузка в векторный регистр



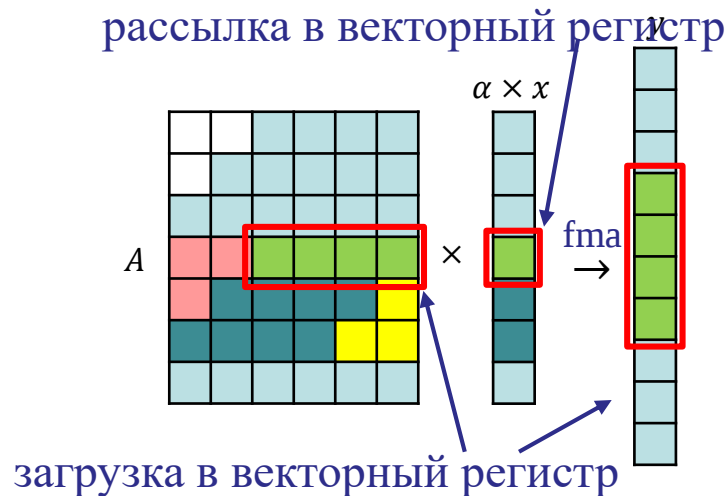
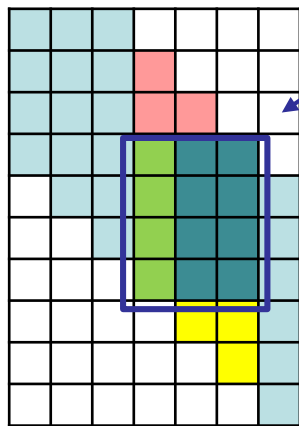
Умножение ленточной матрицы общего вида на вектор (GBMV). Оптимизированный алгоритм 1

```
void GBMV_Optimized(INT m, INT n, INT ku, INT kl, FLOAT alpha, FLOAT *a, INT lda, FLOAT *X, FLOAT *Y){
    INT start = ku, end; ptrdiff_t stride_x, stride_y;
    INT BLOCK_SIZE = GET_VECTOR_LENGTH();
    if (not TRANSPOSED)
        end = MIN(n, m - kl);
    else end = MIN(m, n - kl);
    end -= (end - start) % BLOCK_SIZE;
    /* базовый алгоритм для первых строк */
    if (not TRANSPOSED)
        stride_x = 0; stride_y = - ku + j;
    else
        stride_x = - ku + j; stride_y = 0;
    for (i = start; i + BLOCK_SIZE < end; i += BLOCK_SIZE) { // обход по блокам матрицы
        for (j = 0; j < kl + ku + 1; j++) { // обход по диагоналям в блоке
            x_copy = LOAD(X + i + stride_x); y_copy = LOAD(Y + i + stride_y);
            diag_a = LOAD_WITH_STRIDE(a + j, STRIDE);
            x_copy = MULT(x_copy, factor); y_copy = FMA(diag_a, x_copy, y_copy);
            STORE(Y + i + stride_y, y_copy);
        }
        a += lda * BLOCK_SIZE;
    }
    /* базовый алгоритм для последних строк */
}
```

Умножение ленточной матрицы общего вида на вектор (функция GBMV). Оптимизированный алгоритм 2

- ❑ **Идея:** Матрица разделяется на полосы ширины k , в каждой полосе выделяется плотный прямоугольный блок и два треугольных блока над и под ним. Вычисления для прямоугольного блока выполняются векторно, для треугольных блоков - скалярно.

высота блока = длине векторного регистра



Программная реализация

- ❑ Результаты предварительных экспериментов: производительность функций с ленточными матрицами не зависит от числа строк и столбцов матрицы, но зависит от числа ее диагоналей.
- ❑ Итоговый алгоритм: из общего интерфейса, реализованного в OpenBLAS, выбирается один из оптимизированных алгоритмов или базовый алгоритм в зависимости числа диагоналей матрицы.

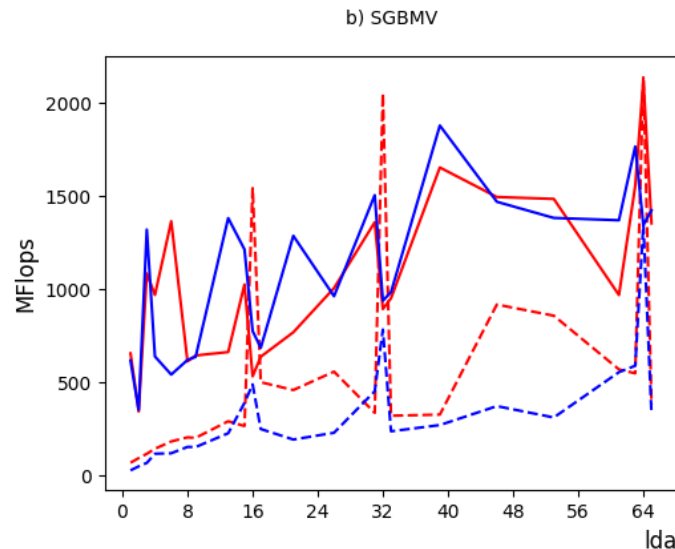
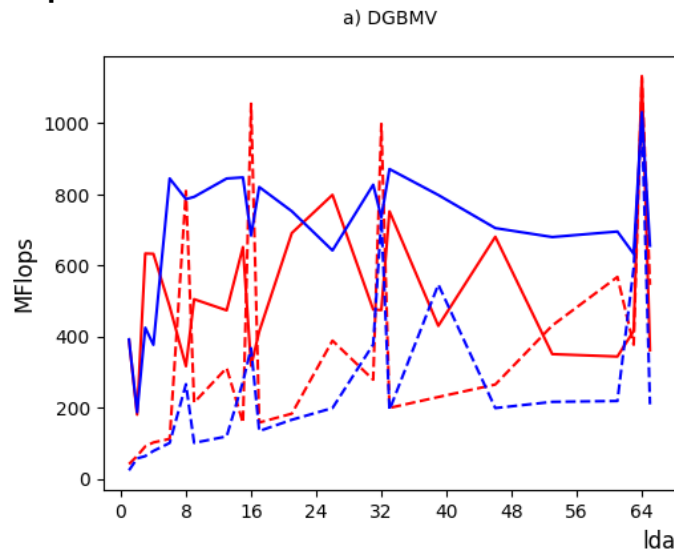
Вычислительная инфраструктура

- ❑ **Плата Lichee Pi 4A**, поддерживает RVV 0.7.1:
 - 4-ядерный процессор T-Head TH1520, частота 1.85GHz, ISA RV64GCV0p7, RAM 16 GB, OS Debian GNU/Linux 12 (bookworm);
 - кросс-компилятор gcc (Xuantie-900 linux-5.10.4 Toolchain V2.8.1 B-20240115) 10.4.0
- ❑ **Плата Banana Pi BPI-F3**, поддерживает RVV 1.0:
 - процессор SpacemiT Keystone K1 с 8 ядрами SpacemiT x60 частотой 1.6GHz, RVA22 Profile, 16 GB оперативной памяти, операционная система Bianbu 1.0.15.
 - кросс-компилятор GCC RISC-V 14.2.0.

Вычислительные эксперименты. GBMV.

Lichee Pi 4A, RVV 0.7.1

Матрица $n = m = 2.5$ млн.



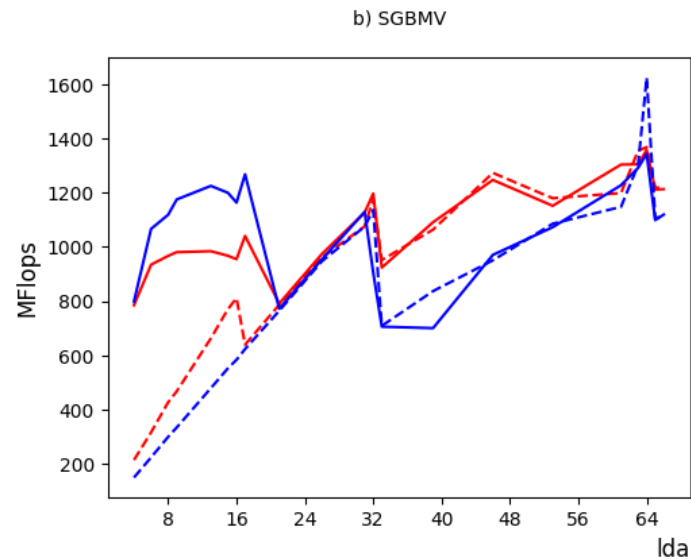
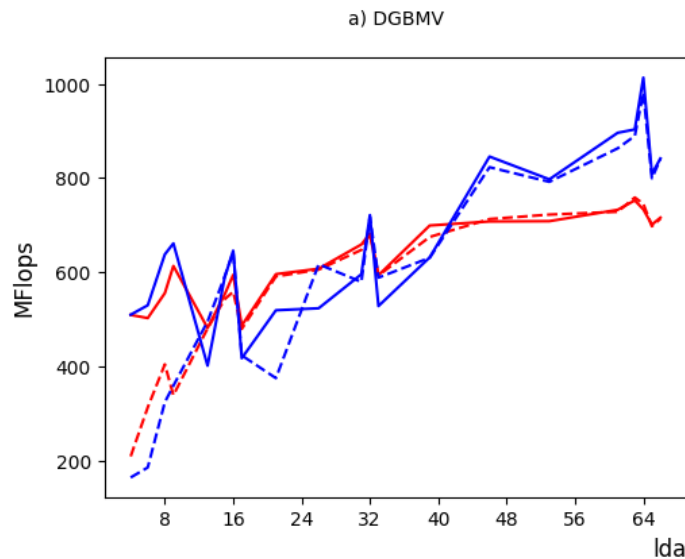
--- reference A — optimized A --- reference A^T — optimized A^T

- Оптимизированный алгоритм **лучше** базовой реализации в **2–7 раз** при $lda < 64$, кроме lda , кратных 8. Оптимизированный алгоритм 1 используется при $lda < 25$ для A^T и $lda < 3$ для A , иначе – алгоритм 2.

Вычислительные эксперименты. GBMV.

Banana Pi BPI-F3, RVV 1.0

Матрица $n = m = 2.5$ млн.



--- reference A — optimized A --- reference A^T — optimized A^T

- Оптимизированный алгоритм **лучше** базовой реализации при $Ida < 9$ для double и $Ida < 17$ для single precision. Среднее опережение – 3 раза.
- Используется оптимизированный алгоритм 1.



TBMV

УМНОЖЕНИЕ ТРЕУГОЛЬНОЙ ЛЕНТОЧНОЙ МАТРИЦЫ НА ВЕКТОР

Умножение треугольной ленточной матрицы на вектор (TBMV)

- ❑ **TBMV**: $x = op(A) \times x$, где x – вектор, A – треугольная ленточная матрица с k побочными диагоналями, размера $n \times n$, $op(A) = A$ или $op(A) = A^T$.
- ❑ Отличия от GBMV:
 - 4 варианта реализации в зависимости от вида треугольника и $op(A)$
 - для нижне-треугольной матрицы обход снизу вверх, для верхне-треугольной – сверху вниз
 - диагональный элемент обрабатывается отдельно (отдельный случай – матрицы с единичной диагональю)
- ❑ Оптимизированные алгоритмы: идеи аналогичны GBMV

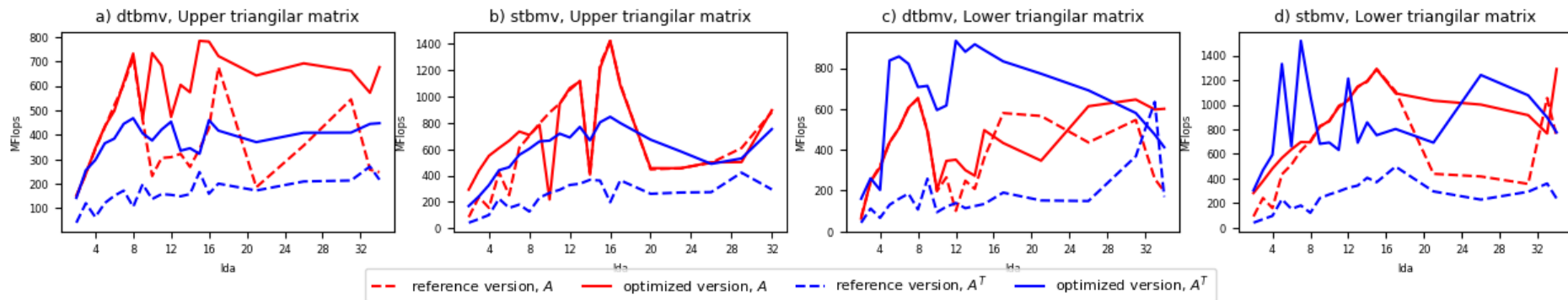
Умножение треугольной ленточной матрицы на вектор (TBMV). Оптимизированный алгоритм 1

```
void TBMV_LN_Optimized(INT n, INT k, FLOAT *a, INT lda, FLOAT *B) {
    INT BLOCK_SIZE = GET_VECTOR_LENGTH();
    a += (n - 1) * lda;
    INT iend = n - k - (n - k) % BLOCK_SIZE;
    for (i = n - 1; i >= iend; i--) { // базовый алгоритм для первых столбцов
        length = MIN(n - i - 1, k);
        if (length > 0) AXPY(length, B[i], a + 1, 1, B + i + 1);
        if (main diagonal is not unit) B[i] *= a[0];
        a -= lda;
    }
    a -= (lda - 1) * BLOCK_SIZE;
    for (; i >= 0; i -= BLOCK_SIZE) { // обход по блокам матрицы
        length = k; b_old = LOAD(B + i);
        if (not a unit main diagonal) { // элемент на главной диагонали обрабатывается отдельно
            diag = LOAD_WITH_STRIDE(a, stride_a);
            z = MUL(b_old, diag); SAVE(B + i, z);
        }
        for (j = 1; j < k + 1; j++) { // обход по диагоналям в блоке
            diag = LOAD_WITH_STRIDE(a + j, stride_a);
            z = LOAD(B + i + j); z = FMA(z, diag, b_old);
            SAVE(B + i + j, z); }
        a -= lda * BLOCK_SIZE;
    } }
}
```

Вычислительные эксперименты. TBMV.

Lichee Pi 4A, RVV 0.7.1

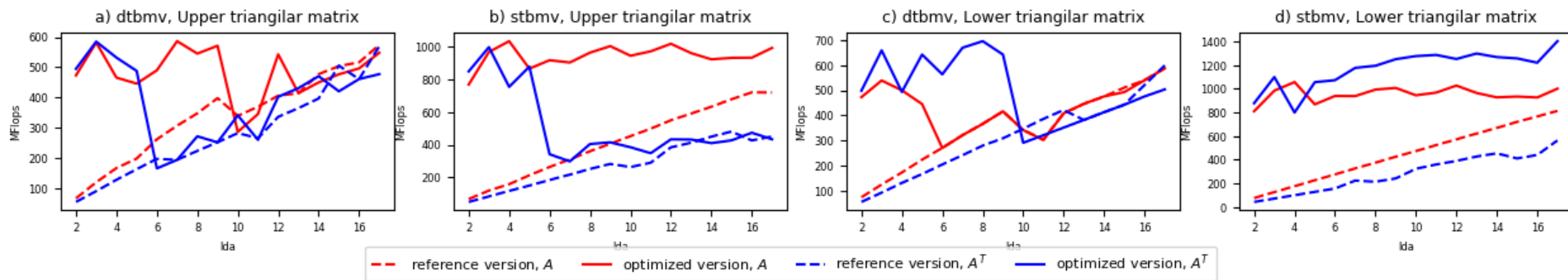
Матрица $n = 1$ млн.



- Производительность и эффективность оптимизации **зависят от варианта функции** (верхний / нижний треугольник, транспонированная или нет матрица)
- Наибольший прирост производительности – на **транспонированных матрицах**, для которых **оптимизированная версия заменяет операцию DOT** в базовом алгоритме. Наименьший – на нижне-треугольных не транспонированных матрицах
- Среднее ускорение – от 1.5 до 5.2 раза.

Вычислительные эксперименты. TBMV. Banana Pi BPI-F3, RVV 1.0

Матрица $n = 1$ млн.



- В сравнении с Lichee Pi 4A: оптимизированная версия опережает базовую на матрицах с меньшей шириной ленты
- Среднее ускорение – от 1.9 до 4.9 раз.



SBMV

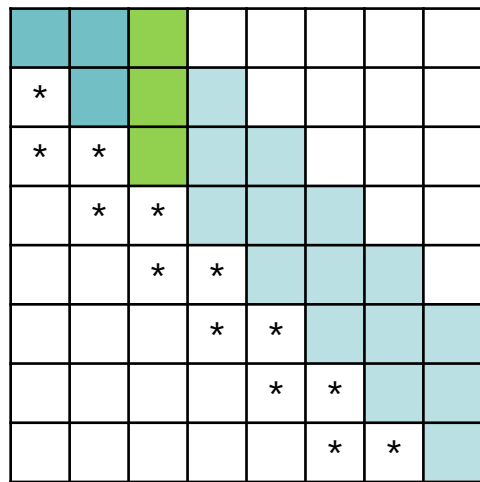
УМНОЖЕНИЕ СИММЕТРИЧНОЙ ЛЕНТОЧНОЙ МАТРИЦЫ НА ВЕКТОР

Умножение симметричной ленточной матрицы на вектор (SBMV). Базовый алгоритм

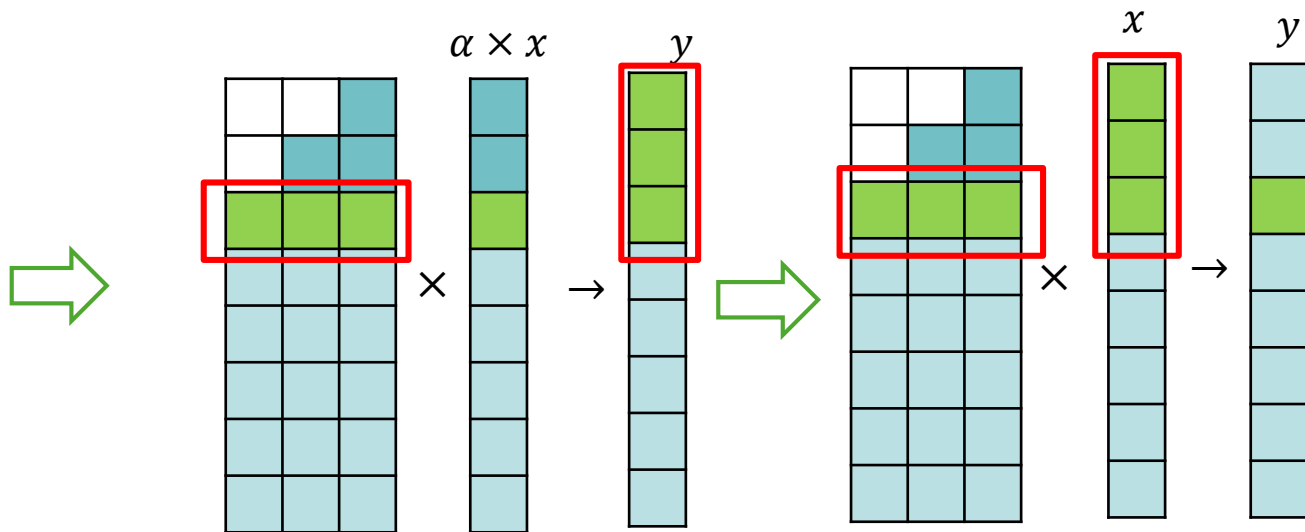
- ❑ **SBMV:** $y = \alpha \times A \times x + \beta \times y$, где x, y – вектора, α, β – скаляры, A – симметричная ленточная матрица с k побочными диагоналями, размера $n \times n$.
- ❑ Отличие от GBMV: На каждой итерации вычисляется произведение для одного столбца матрицы A , вызываются обе функции: AXPY и DOT.

```
void SBMV(INT n, INT k, FLOAT alpha, FLOAT *a, INT lda, FLOAT *X, FLOAT *Y) {  
    for (i = 0; i < n; i++) {  
        #ifndef LOWER  
            length = MIN(i, k);  
            AXPYU_K(length + 1, alpha * X[i], a + k - length, Y + i - length);  
            Y[i] += alpha * DOTU_K(length, a + k - length, 1, X + i - length);  
        #else  
            length = MIN(k, n - i - 1);  
            AXPYU(length + 1, alpha * X[i], a, Y + i);  
            Y[i] += alpha * DOTU_K(length, a + 1, 1, X + i + 1, 1);  
        #endif  
        a += lda;  
    }  
}
```

Умножение симметричной ленточной матрицы на вектор (функция SBMV). Базовый алгоритм



* – здесь есть ненулевые
элементы, но они не
хранятся

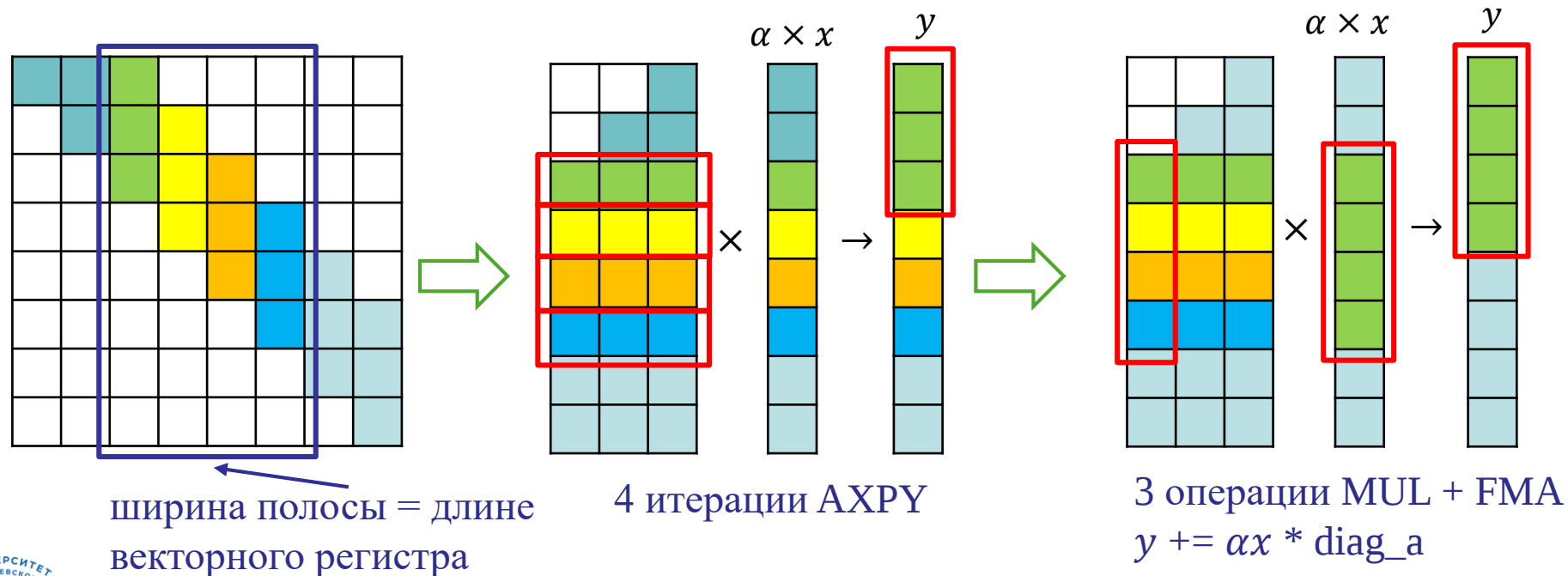


```
AXPY(length + 1,  
alpha * X[i],  
a + k - length,  
Y + i - length);
```

```
Y[i] += alpha *  
DOT(length,  
a + k - length,  
X + i - length);
```

Умножение симметричной ленточной матрицы на вектор (функция SBMV). Оптимизированный алгоритм 1

- Идея: все вызовы DOT заменены на обход матрицы по диагоналям, оставлены вызовы AXPY.



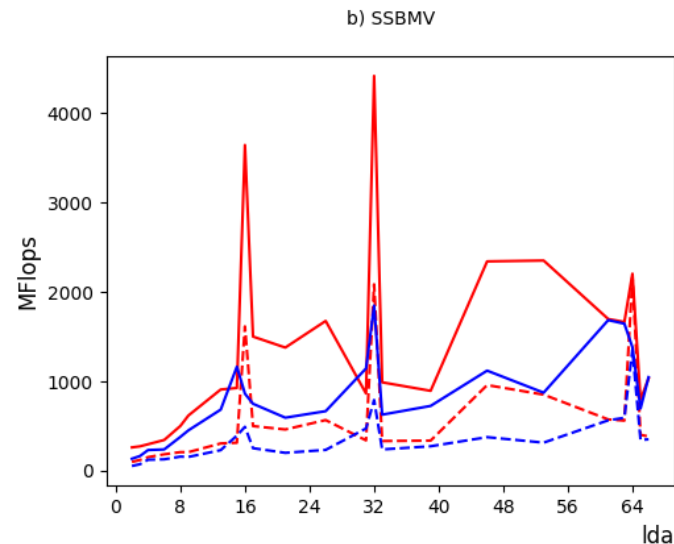
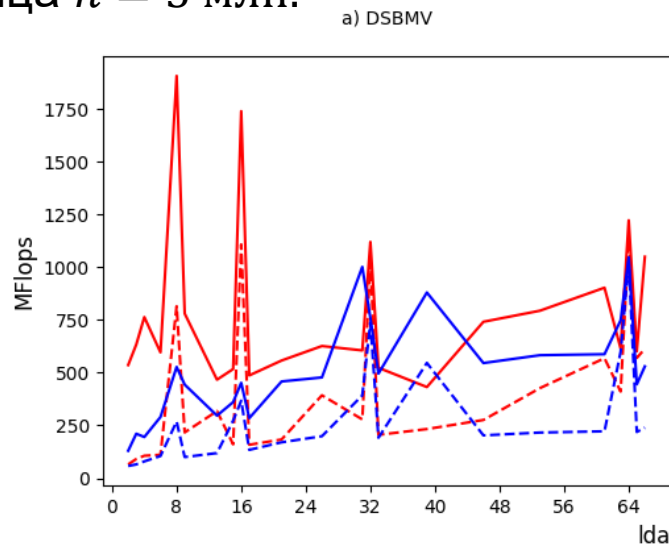
Умножение симметричной ленточной матрицы на вектор (SBMV). Оптимизированный алгоритм 1

```
void SBMV_L_Optimized(INT n, INT k, FLOAT alpha, FLOAT *a,
                      INT lda, FLOAT *X, FLOAT *Y) {
    INT BLOCK_SIZE = GET_VECTOR_LENGTH(), iend = (n - k) - (n - k) % BLOCK_SIZE;
    for (i = 0; i < n - k; i++) { // AXPY для всех столбцов с lda элементами
        length = MIN(k, n - i - 1); AXPY(length + 1, alpha * X[i], a + lda * i, Y + i);
    }
    for (i = 0; i < iend; i += BLOCK_SIZE) { // обход матрицы по блокам
        y_copy = LOAD(y + i);
        for (INT j = 0; j < k; j++) { // обход по диагоналям в блоке
            x_copy = LOAD(x + i + 1 + j); diag_a = LOAD_WITH_STRIDE(a + 1 + j, STRIDE);
            mul = MUL(x_copy, diag_a); y_copy = FMA(y_copy, alpha, mul);
        }
        SAVE(y + i, y_copy); a += BLOCK_SIZE * lda;
    }
    for (; i < n - k; i++) { // DOT для столбцов с lda элементами, не вошедших в блоки
        length = MIN(n - i - 1, k); Y[i] += alpha * DOT(length, a + 1, X + i + 1); a += lda;
    }
    for (; i < n; i++) { // базовый алгоритм для последних строк с < lda элементов
        length = MIN(n - i - 1, k);
        AXPY(length + 1, alpha * X[i], a + k - length, Y + i - length);
        Y[i] += alpha * DOT(length, a + 1, X + i + 1); a += lda;
    }
}
```

Вычислительные эксперименты. SBMV.

Lichee Pi 4A, RVV 0.7.1

Матрица $n = 5$ млн.



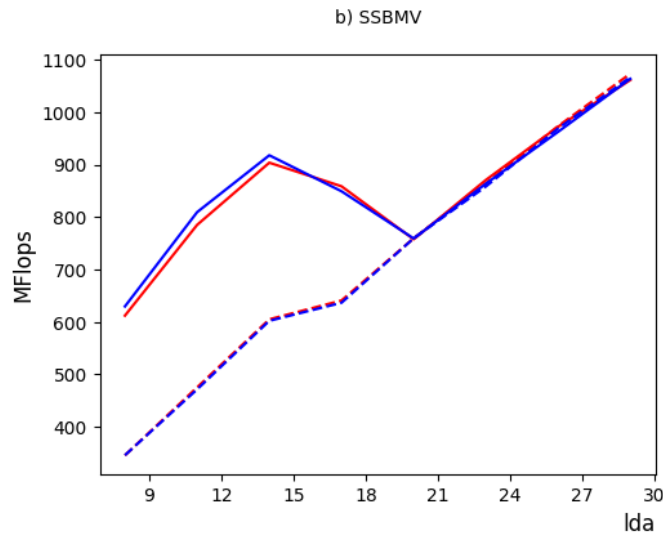
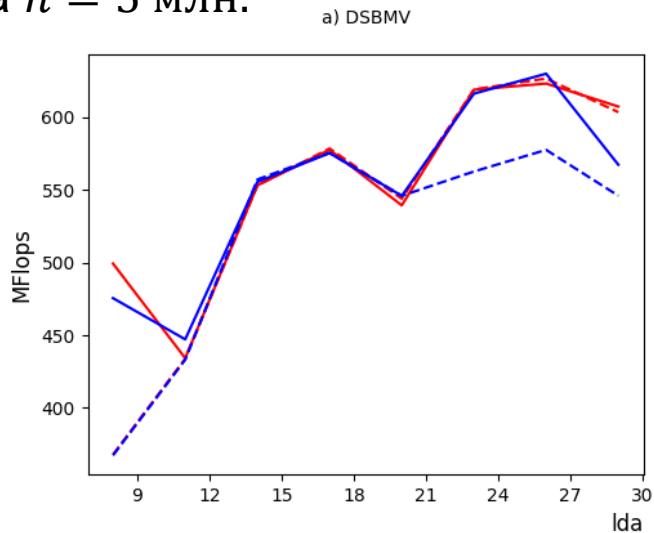
--- reference version, L — optimized version, L --- reference version, U — optimized version, U

- Оптимизированный алгоритм **2 лучше** базовой реализации **в 1.5-3 раза**, кроме lda, кратных 32 и 64.

Вычислительные эксперименты. SBMV.

Banana Pi BPI-F3, RVV 1.0

Матрица $n = 5$ млн.



--- reference version, L — optimized version, L --- reference version, U — optimized version, U

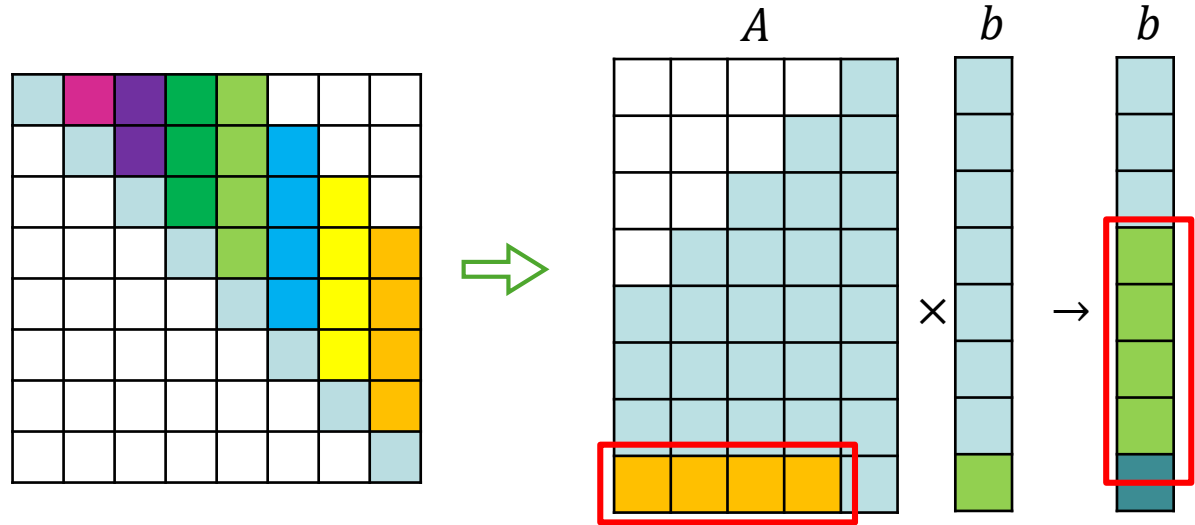
- Оптимизированный алгоритм 1 используется при малом числе диагоналей, оптимизированный алгоритм 2 – при $Ida > 14$ для double, $Ida > 20$ для float.
- Среднее ускорение – **1.6** раз

TBSV

РЕШЕНИЕ СЛАУ С ТРЕУГОЛЬНОЙ ЛЕНТОЧНОЙ МАТРИЦЕЙ

Решение СЛАУ с треугольной ленточной матрицей (TBSV)

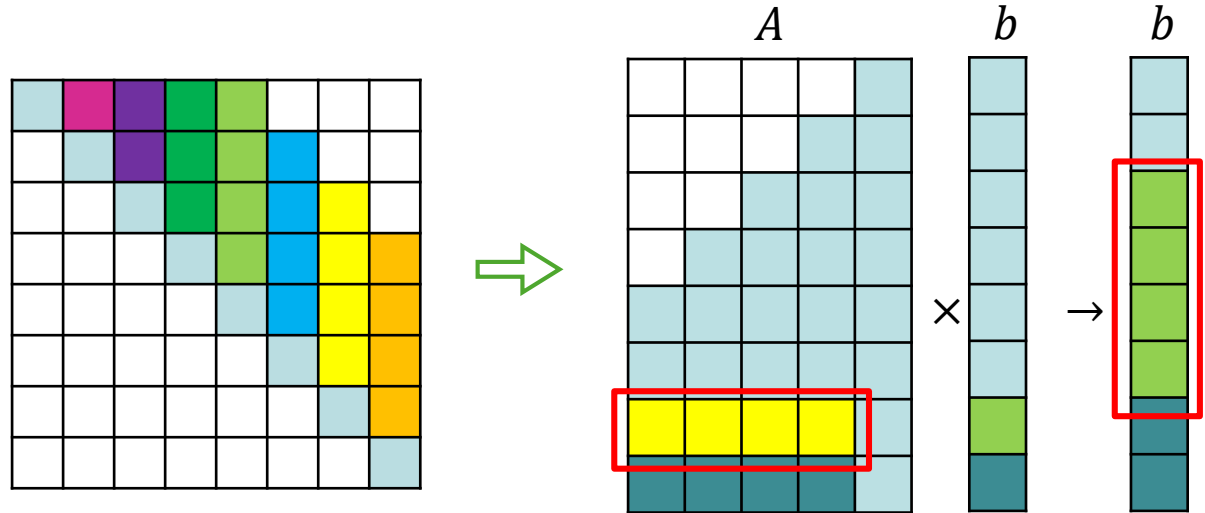
- Функция TBSV: решение СЛАУ $op(A)x = b$, где x, b – вектора, A – треугольная ленточная матрица с k побочными диагоналями, размера $n \times n$, $op(A) = A$ или $op(A) = A^T$.



```
B[i] -= DOT(length, a + k - length, B + i - length);
```

Решение СЛАУ с треугольной ленточной матрицей (TBSV)

- Функция TBSV: решение СЛАУ $op(A)x = b$, где x, b – вектора, A – треугольная ленточная матрица с k побочными диагоналями, размера $n \times n$, $op(A) = A$ или $op(A) = A^T$.



```
B[i] -= DOT(length, a + k - length, B + i - length);
```

Решение СЛАУ с треугольной ленточной матрицей (TBSV). Базовый алгоритм

```
void TBSV_L(INT n, INT k, FLOAT *a, INT lda, FLOAT *B){
    INT i, length;
    for (i = 0; i < n; i++) {
        if (TRANPOSED) {
            length = MIN(i, k);
            if (length > 0) {
                B[i] -= DOT(length, a + k - length, B + i - length);
            }
            B[i] /= a[k];
        }
        else {
            B[i] /= a[0];
            length = MIN(n - i - 1, k);
            if (length > 0) {
                AXPY(length, -B[i], a + 1, B + i + 1);
            }
            a += lda;
        }
    }
}
```

- ❑ Идея оптимизации: своя реализация AXPY и DOT

Решение СЛАУ с треугольной ленточной матрицей (TBSV). Оптимизированный алгоритм...

```
void TBSV_LN_Optimized(INT n, INT k, FLOAT *a, INT lda, FLOAT *B) {  
    for (i = 0; i <= n - k - 1; i++) {  
        B[i] /= a[0];  
        // вариант реализации AXPY  
        A_ptr = a + 1;  
        B_ptr = B + i + 1;  
        for (j = k; j > 0; ) {  
            a_copy = LOAD(A_ptr);  
            b_copy = LOAD(B_ptr);  
            mult = BROADCAST(-B[i]);  
            b_copy = FMA(mult, a_copy, b_copy);  
            STORE(B_ptr, b_copy);  
            step = min(j, MAX_VECTOR_LENGTH);  
            A_ptr += step; B_ptr += step; j -= step;  
            a += lda;  
        }  
        /* базовый алгоритм для последних k строк */  
    }  
}
```

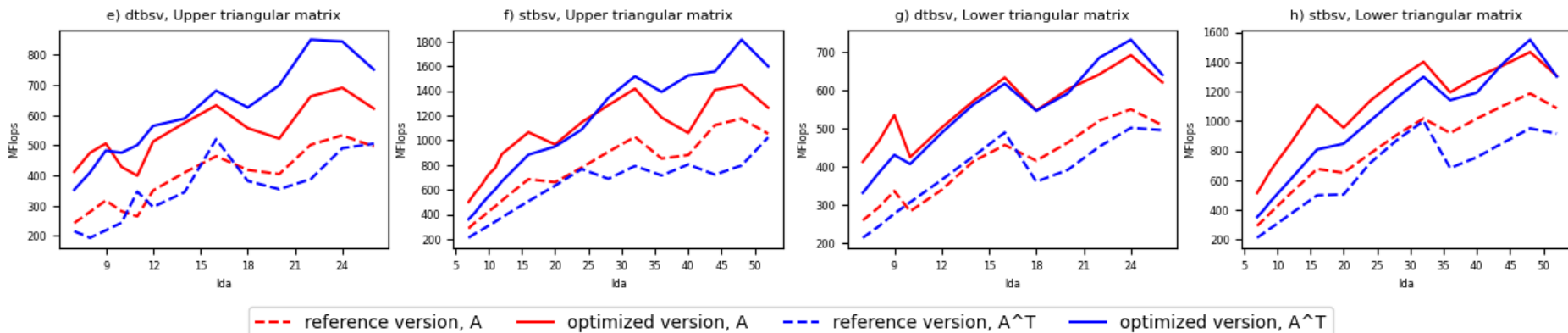
Решение СЛАУ с треугольной ленточной матрицей (TBSV). Оптимизированный алгоритм...

```
void TBSV_LT_Optimized(INT n, INT k, FLOAT *a, INT lda, FLOAT *B) {  
    /* базовый алгоритм для первых k строк */  
    for (i = k; i < n; i++) {  
        // вариант реализации DOT  
        A_ptr = a;  
        B_ptr = B + i - k;  
        vector_sum = BROADCAST(0);  
        for (int j = k; j > 0; j++) {  
            a_copy = LOAD(A_ptr);  
            b_copy = LOAD(B_ptr);  
            vector_sum = FMA(a_copy, b_copy, vector_sum);  
            step = min(j, MAX_VECTOR_LENGTH);  
            A_ptr += step; B_ptr += step; j -= step;  
        }  
        dot = SUM(vector_sum);  
        B[i] = (B[i] - dot) / a[k];  
        A += lda;  
    }  
}
```

Вычислительные эксперименты. TBSV.

Lichee Pi 4A, RVV 0.7.1

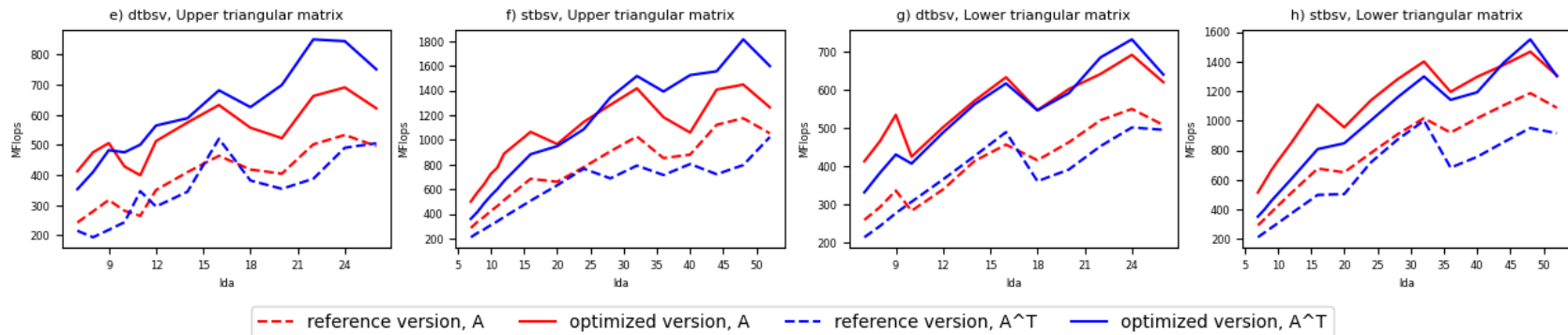
Матрица $n = 500\,000$



- Производительность зависит от вида треугольника и транспонированности матрицы.
- Для верхне-треугольной матрицы оптимизированный алгоритм **лучше** базовой реализации при «среднем» lda в 1.1 раз, для транспонированной верхне-треугольной матрицы – при всех lda, **в 2–3 раза**.
- Для нижне-треугольной нетранспонированной матрицы оптимизированный алгоритм **лучше** базовой реализации в 2 раза, транспонированной – **в 1.3 раза**.

Вычислительные эксперименты. TBSV. Banana Pi BPI-F3, RVV 1.0

$n = 500\,000$



- Для верхне- и нижне-треугольных матриц оптимизированный алгоритм **лучше** базовой реализации при всех lda , в среднем, **в 1.6 раз**.

Заключение

- ❑ Был предложен подход к оптимизации алгоритмов умножения ленточных матриц на вектор, в основе которого – изменение порядка обхода матриц.
- ❑ Эффективность предложенных алгоритмов зависит от числа диагоналей матрицы.
- ❑ Использование векторных инструкций позволяет значительно ускорить вычисления. Однако реализация с использованием интринсиков не переносима на другие архитектуры, должна разрабатываться отдельно под каждый новый набор векторных команд.
- ❑ Приемы повышения производительности, работающие для матрично-векторных операций на процессорах x86, также работают и на процессорах RISC-V.

Программная реализация алгоритмов выполнена Воденеевой А.А., Ковалевым К.И., Устиновым А.В., Козиновым Е.А., Пировой А.Ю.

Литература

1. Goto, K., & Geijn, R. A. V. D. (2008). Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS), 34(3), 1-25.
2. Goto, K., & Van De Geijn, R. (2008). High-performance implementation of the level-3 BLAS. ACM Transactions on Mathematical Software (TOMS), 35(1), 1-14.
3. Remón, A., Quintana-Ortí, E. S., & Quintana-Ortí, G. (2007). The implementation of BLAS for band matrices. In International Conference on Parallel Processing and Applied Mathematics (pp. 668-677). Berlin, Heidelberg: Springer Berlin Heidelberg.
4. OpenBLAS <https://github.com/OpenMathLib/OpenBLAS>

Контакты

Нижегородский государственный университет

<http://www.unn.ru>

Институт информационных технологий, математики и механики

<http://www.itmm.unn.ru>

Кафедра высокопроизводительных вычислений и системного программирования

