Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

Об опыте оптимизации базовых алгоритмов работы с ленточными матрицами в библиотеке OpenBLAS

Пирова А.Ю., Воденеева А.А.,
Ковалев К.И., Устинов А.В.,
Козинов Е.А., Линев А.В.,
Волокитин В.В., Мееров И.Б.

«Суперкомпьютерные дни в России»
Москва, 29-30 сентября 2025 г.

Реализации стандарта BLAS

- **BLAS** (Basic Linear Algebra Subprograms) – интерфейс библиотек для базовых операций линейной алгебры (Lawson, Hanson at el., 1979)
- Реализации стандарта BLAS 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
- **Потенциал для оптимизации есть (почти) всегда**

OpenBLAS, реализация на RISC-V

- BLAS level 1 – низкоуровневые реализации под различные архитектуры (интринсики, ассемблер) большинства функций;
- BLAS level 2 – низкоуровневые реализации ядер для некоторых функций (GEMV, SYMV), остальные вызывают BLAS level 1;
- BLAS level 3 – низкоуровневые реализации ядер для некоторых функций (GEMM, TRSM, TRMM), остальные вызывают низкоуровневые ядра для функций BLAS level 1, 2, 3.

Выбор алгоритмов

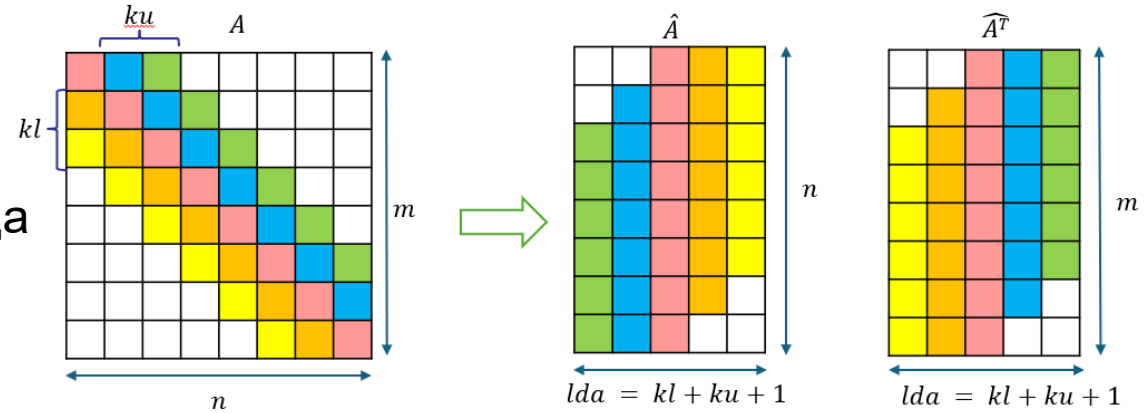
- Разработана **система тестирования** корректности и производительности для алгоритмов BLAS на основе библиотеки BLAS Tester.
- Выбраны тестовые данные (сгенерированные матрицы). Матрицы разбиты на «маленькие», «средние», «большие» по объему памяти для хранения
- Проведены масштабные эксперименты на суперкомпьютере (Intel x86) для сравнения производительности **OpenBLAS и MKL**.
- Обнаружено, что работа с ленточными матрицами организована в OpenBLAS **недостаточно эффективно**.

Цель работы

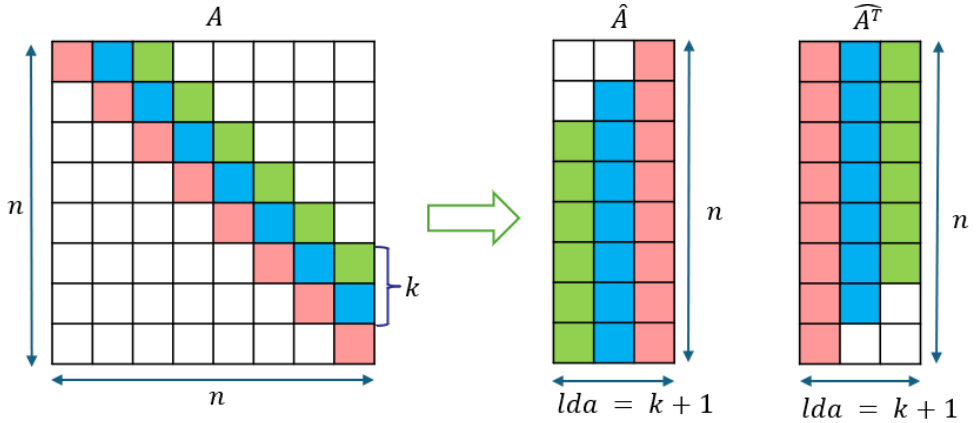
- Цель работы - оптимизация реализаций некоторых матрично-векторных операций с ленточными матрицами в библиотеке OpenBLAS для процессоров архитектуры RISC-V.
- По результатам тестирования выбраны четыре функции BLAS-2:
  - **GBMV** – умножение ленточной матрицы общего вида на вектор;
  - **SBMV** – умножение симметричной ленточной матрицы на вектор;
  - **TBMV** – умножение треугольной ленточной матрицы на вектор;
  - **TBSV** – решение СЛАУ с треугольной ленточной матрицей.

Формат хранения

Ленточная матрица общего вида



Треугольная или симметричная ленточная матрица



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 $op(A) = A^T$.

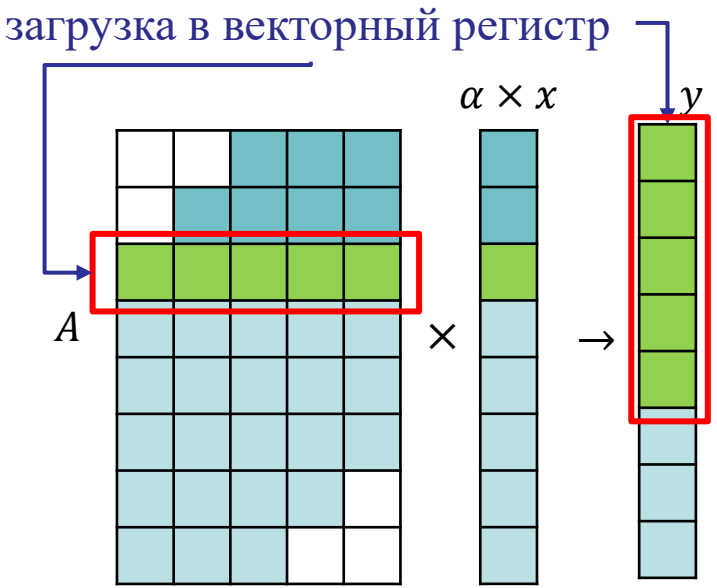
- **Не транспонированная матрица**

  Основная вычислительная операция - **AXPY**: $y = \alpha \times x + y$,
  где x, y – векторы, α – скаляр

- **Транспонированная матрица**

  Основная вычислительная операция – **DOT**: $res = \sum_{i=1}^{n} x_i y_i$, x, y – векторы

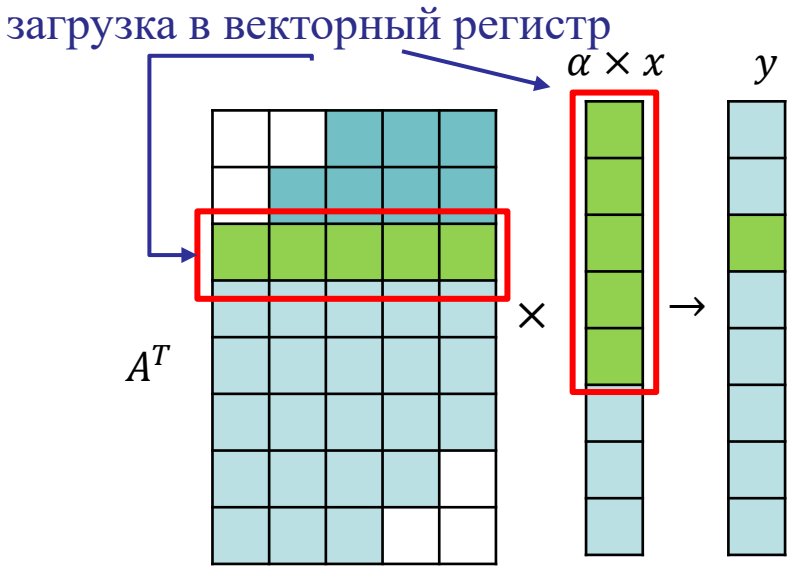
- В OpenBLAS векторизация выполнена в AXPY и DOT



Умножение ленточной матрицы общего вида на вектор (GBMV)

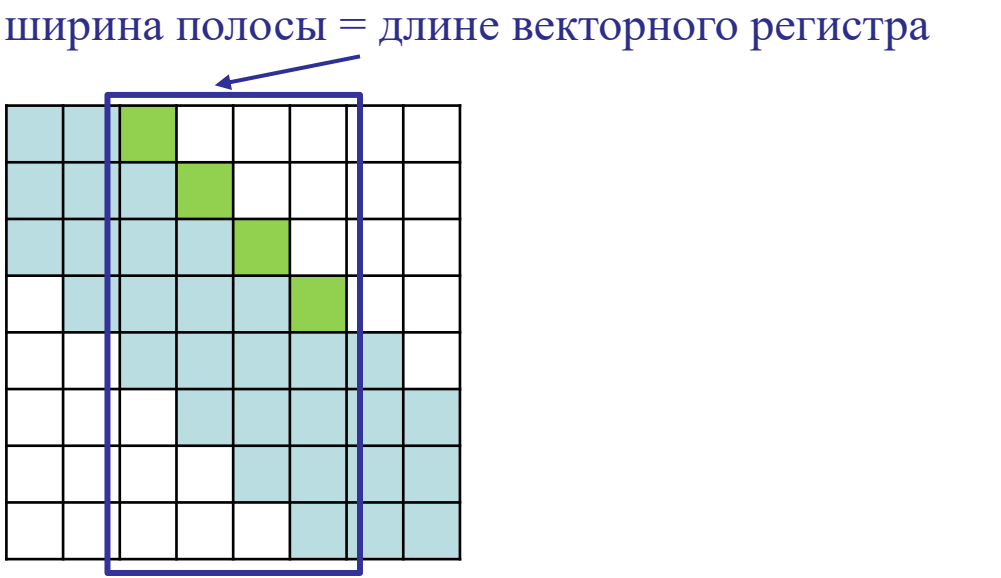
- Функция GBMV: $y = \alpha \times op(A) \times x + \beta \times y$, где x, y – вектора, α, β – скаляры, A – ленточная матрица с kl нижними и ku верхними диагоналями, размера $m \times n$, $op(A) = A$ или $op(A) = A^T$.

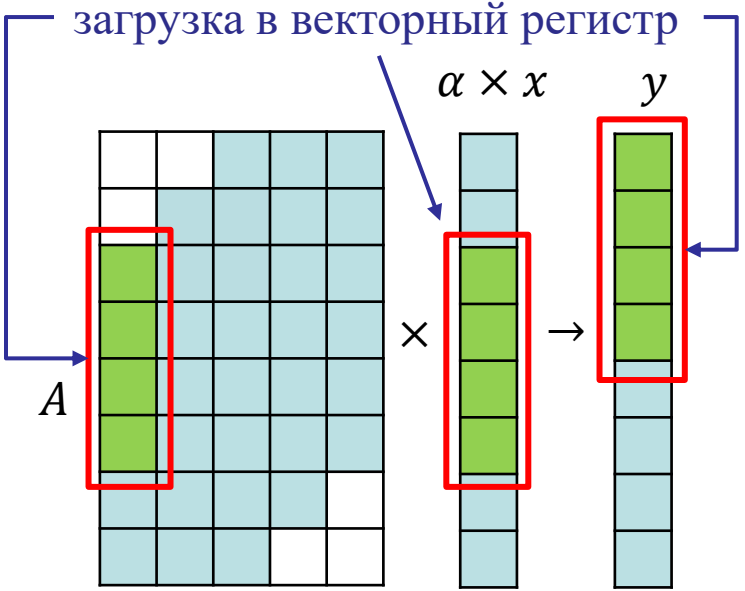
- В OpenBLAS векторизация выполнена в AXPY и DOT
- **Проблемы производительности:**
  - при малом числе диагоналей вызываются скалярные версии AXPY и DOT;
  - при большом числе диагоналей можно лучше использовать подсистему памяти.



Алгоритм умножения ленточной матрицы общего вида на вектор (GBMV) для матриц с малым числом диагоналей

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

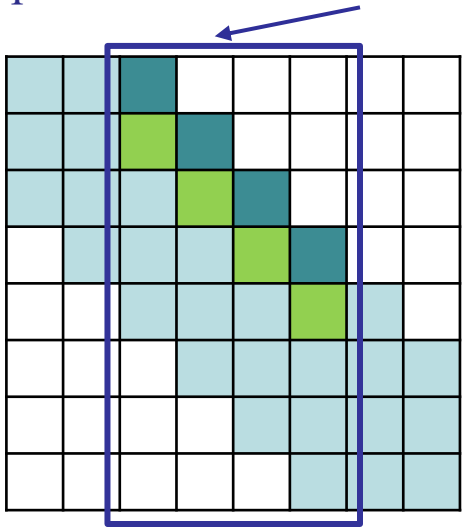




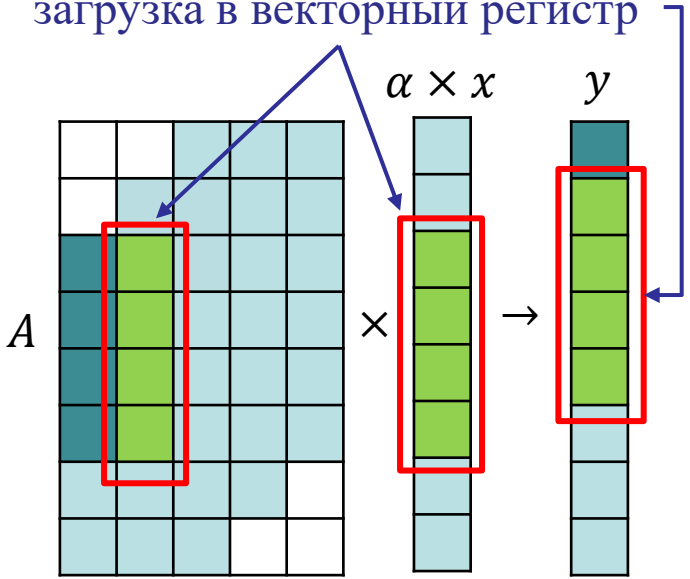
Алгоритм умножения ленточной матрицы общего вида на вектор (GBMV) для матриц с малым числом диагоналей

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

ширина полосы = длине векторного регистра

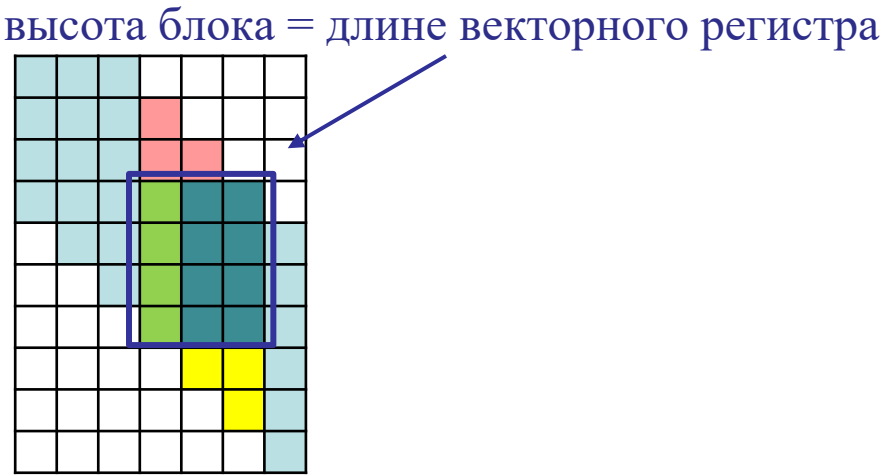


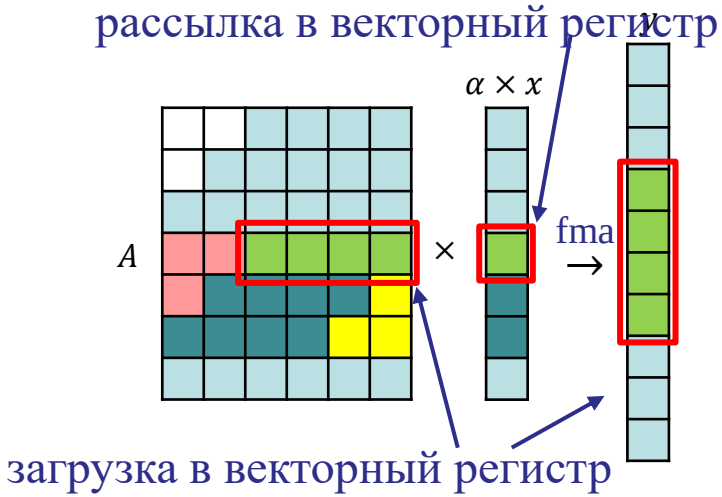
загрузка в векторный регистр



Алгоритм умножения ленточной матрицы общего вида на вектор (GBMV) для матриц с большим числом диагоналей

- **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.



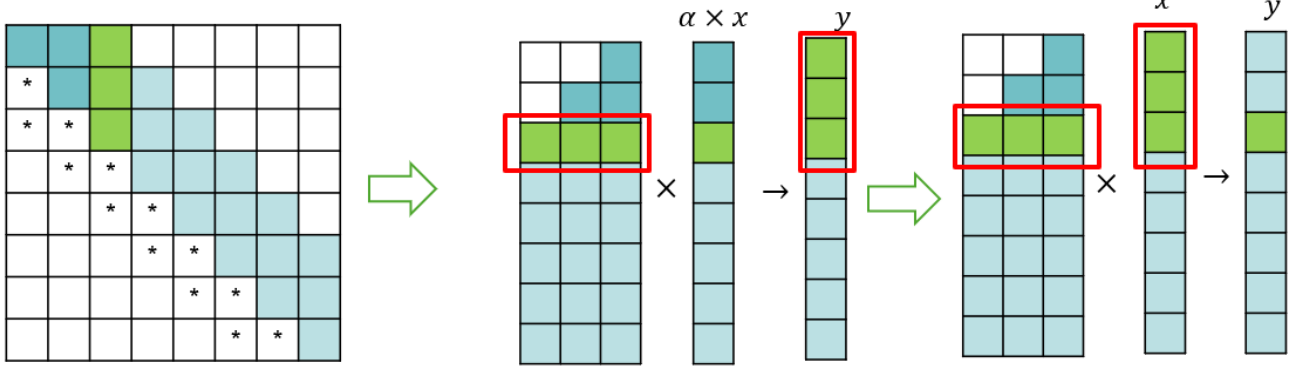


Умножение треугольной ленточной матрицы на вектор (TBMV)

- **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 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.



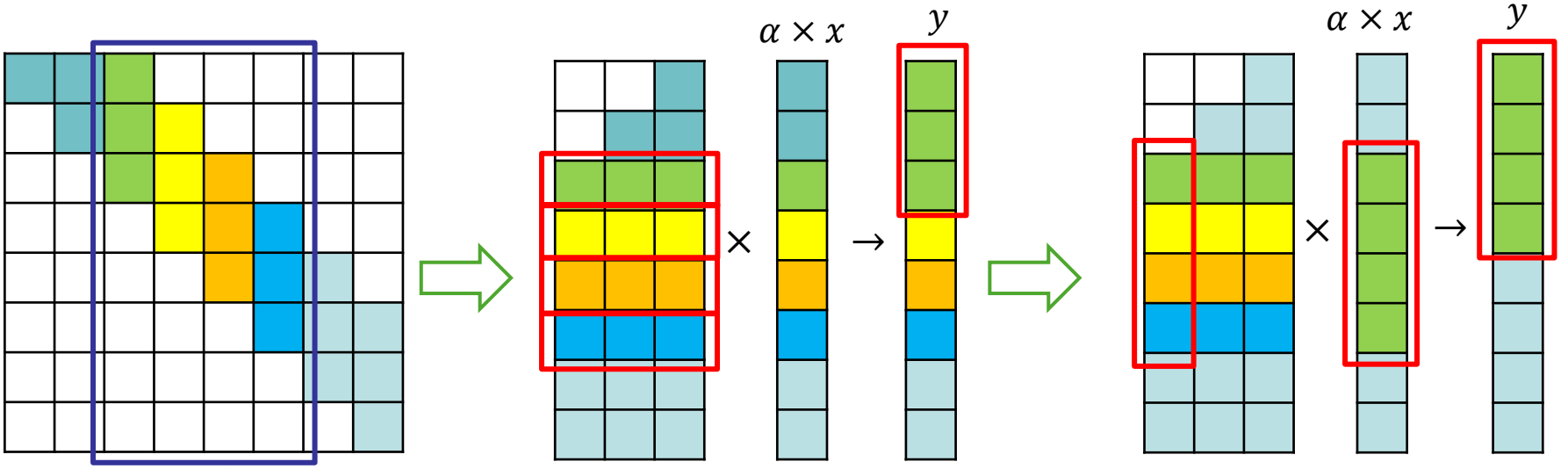
* – здесь есть ненулевые элементы, но они не хранятся

```
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) для матриц с малым числом диагоналей

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



ширина полосы = длине векторного регистра

4 итерации AXPY

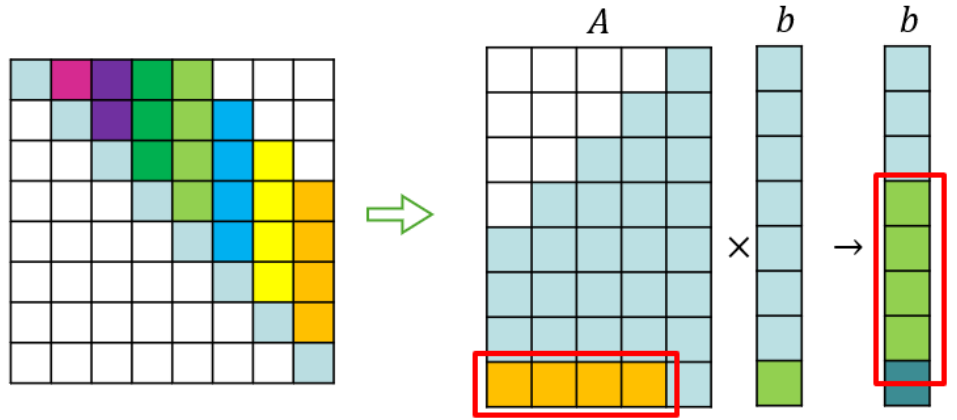
3 операции MUL + FMA

y += αx * diag_a

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 $op(A) = A^T$.

- **Базовая реализация**:
  на каждой итерации выполняется векторная операция AXPY или DOT
- **Идея оптимизации**:
  своя реализация AXPY и DOT




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

Программная реализация

- Базовая версия – библиотека OpenBLAS.
- В библиотеку интегрированы реализации рассмотренных алгоритмов для матриц с малым и большим числом диагоналей, векторизованные для процессоров RISC-V.
  - наборы инструкций RVV 0.7.1, RVV 1.0; наилучший LMUL подобран экспериментально;
  - точность одинарная и двойная;
  - последовательная версия.
- Итоговый алгоритм для функций GBMV, SBMV, TBMV:
  из общего интерфейса, реализованного в OpenBLAS, выбирается один из оптимизированных алгоритмов или базовый алгоритм, в зависимости числа диагоналей матрицы. Пороги переключения подобраны экспериментально.
- Код доступен:
  https://github.com/UNN-ITMM-Software/OpenBLAS/tree/band_matrix_RVV_improving

Вычислительная инфраструктура

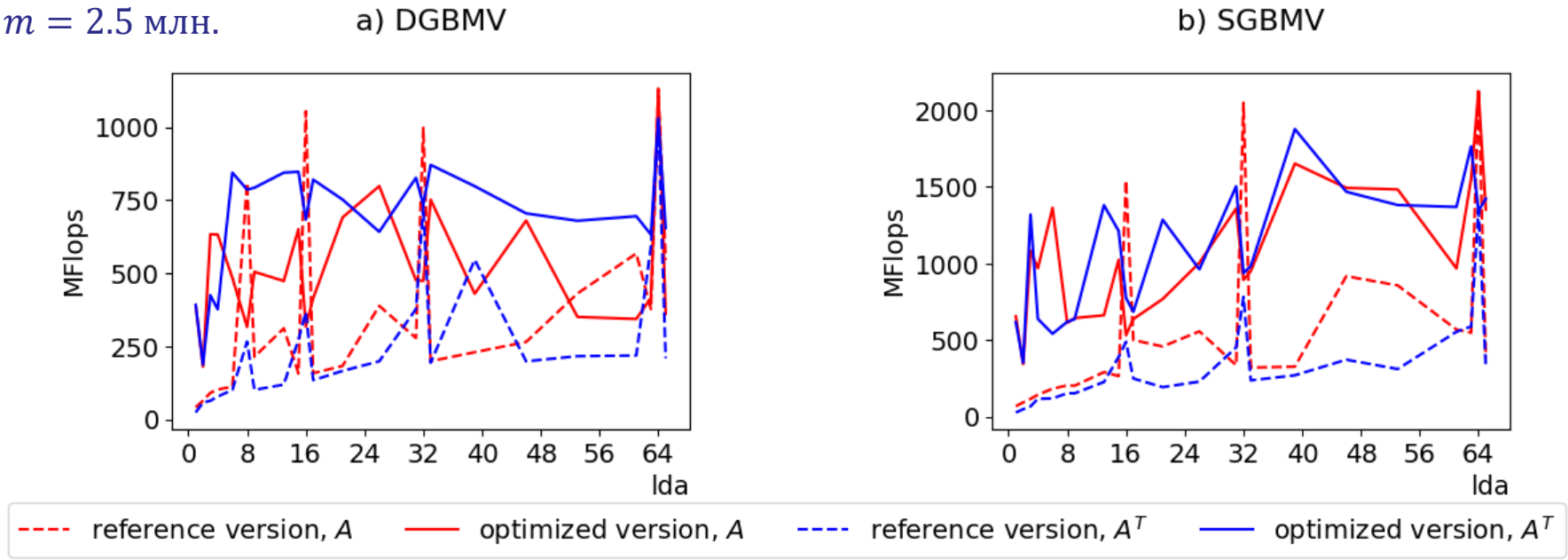
- **Плата Lichee Pi 4A**, поддерживает RVV 0.7.1:
  - 4-ядерный процессор T-Head TH1520, частота 1.85GHz, ISA RV64GCV0p7, RAM 16 GB, OS Debian GNU/Linux 12 (bookworm);
  - кросс-компилятор gcc (Xuantie-900 linux-5.10.4 Toolchain V2.8.1 B-20240115) 10.4.0
- **Плата Banana Pi BPI-F3**, поддерживает RVV 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.

Обозначения:

- алгоритм 1 – алгоритм для матрицы с малым числом диагоналей,
- алгоритм 2 – алгоритм для матриц с большим числом диагоналей,
- lda – число ненулевых диагоналей матрицы.

Вычислительные эксперименты. GBMV. Lichee Pi 4A, RVV 0.7.1

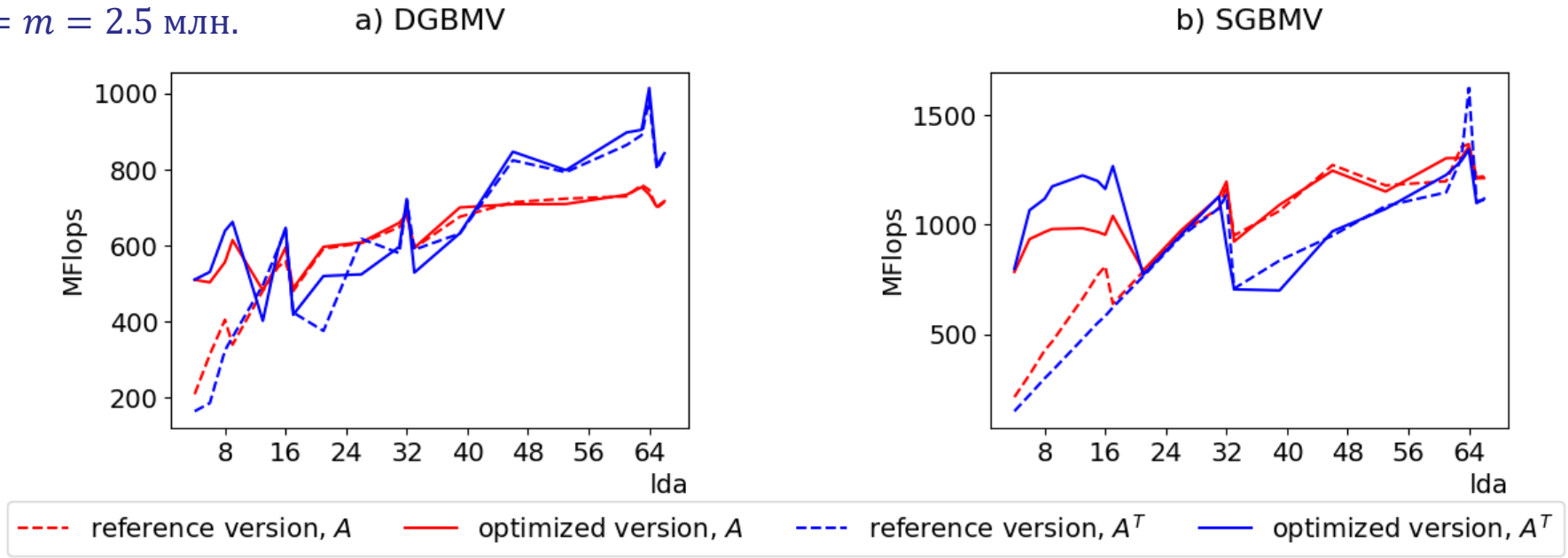
$n = m = 2.5$ млн.



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

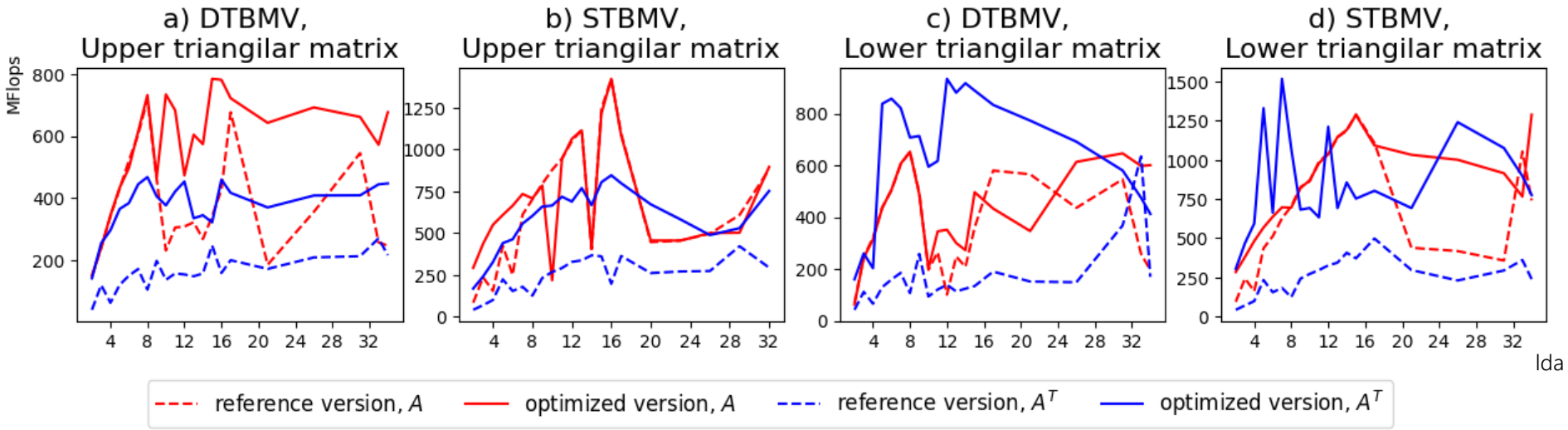
Вычислительные эксперименты. GBMV. Banana Pi BPI-F3, RVV 1.0

$n = m = 2.5$ млн.



- Предложенный алгоритм 1 **лучше** базовой реализации при **lda < 9** для double и **lda < 17** для single precision. Среднее опережение – **3 раза**.
- Используется алгоритм 1.

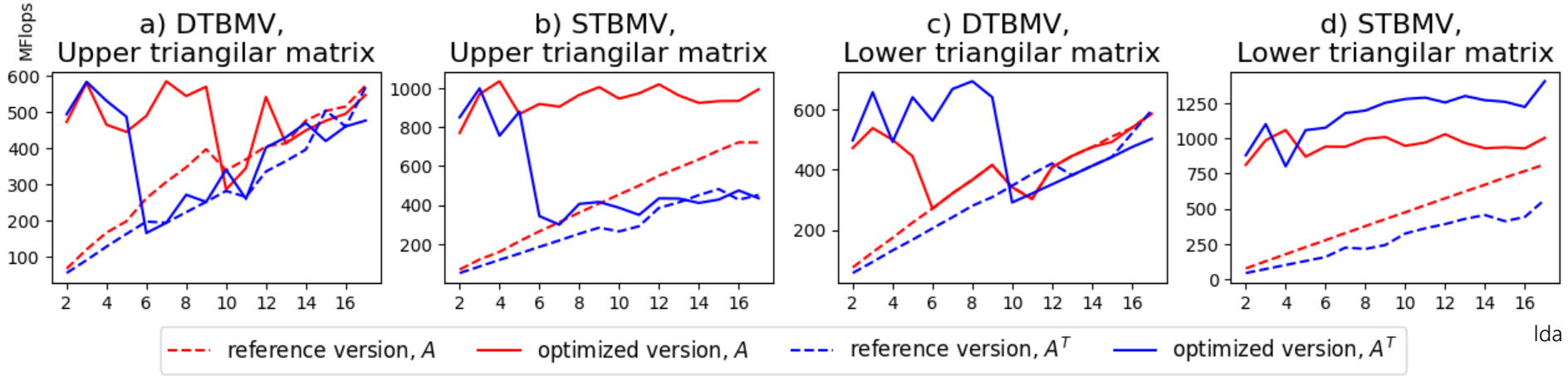
Вычислительные эксперименты. TBMV. Lichee Pi 4A, RVV 0.7.1



- 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.

$n = 1$ млн.

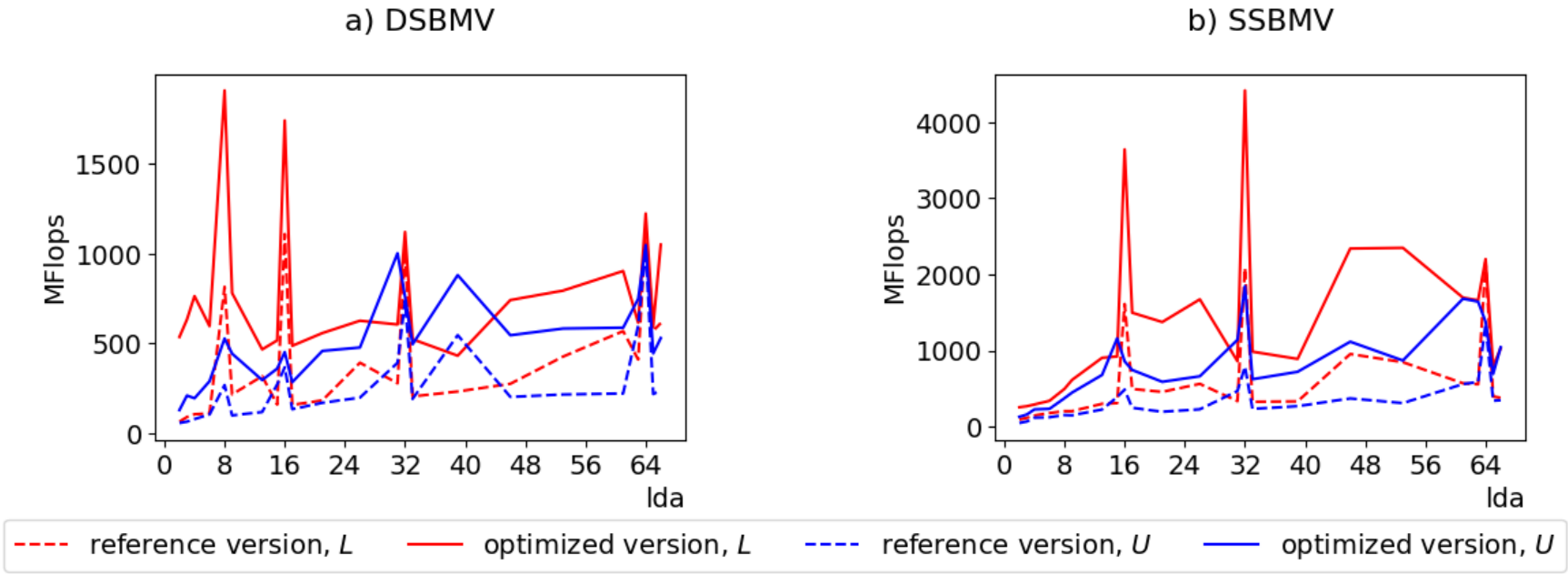
Вычислительные эксперименты. TBMV. Banana Pi BPI-F3, RVV 1.0



- В сравнении с Lichee Pi 4A: новые алгоритмы опережают базовую реализацию на матрицах с меньшей шириной ленты.
- Среднее ускорение – **от 1.9 до 4.9** раз.

$n = 1$ млн.

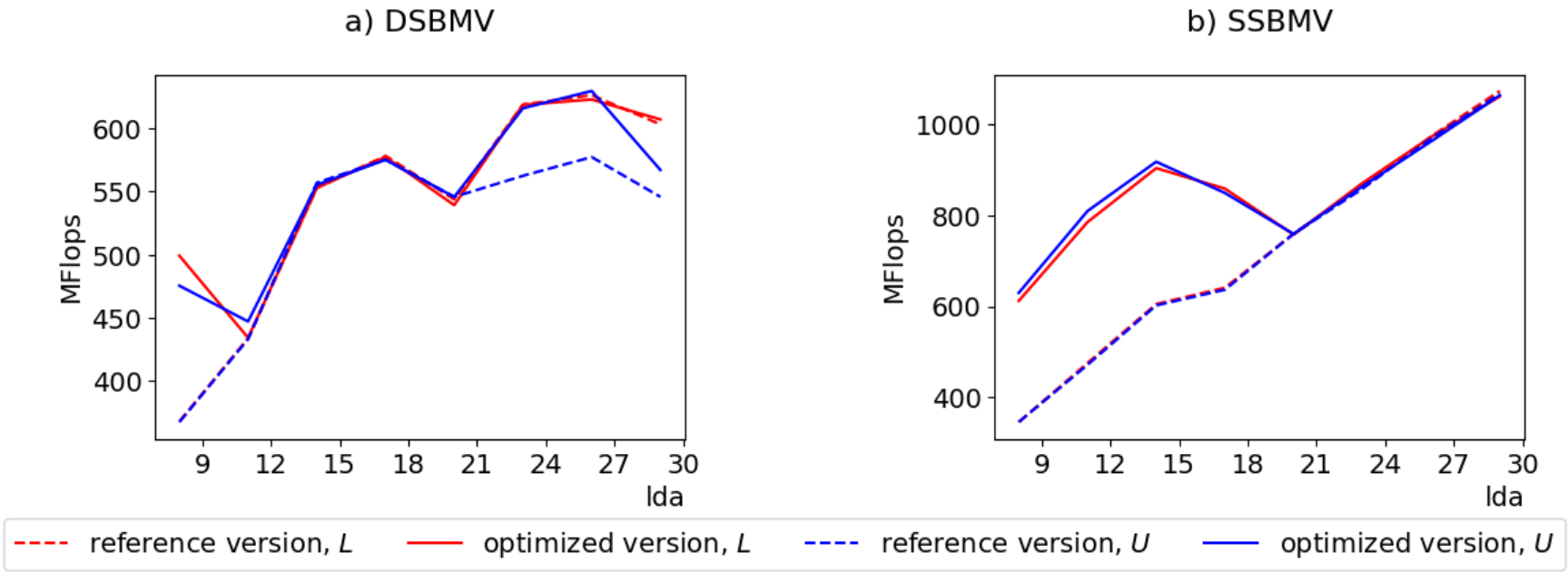
Вычислительные эксперименты. SBMV. Lichee Pi 4A, RVV 0.7.1



- Алгоритм 2 всегда лучше алгоритма 1, а также **лучше** базовой реализации **в 1.5-3 раза**, кроме lda, кратных 32 и 64.

$n = 5$ млн.

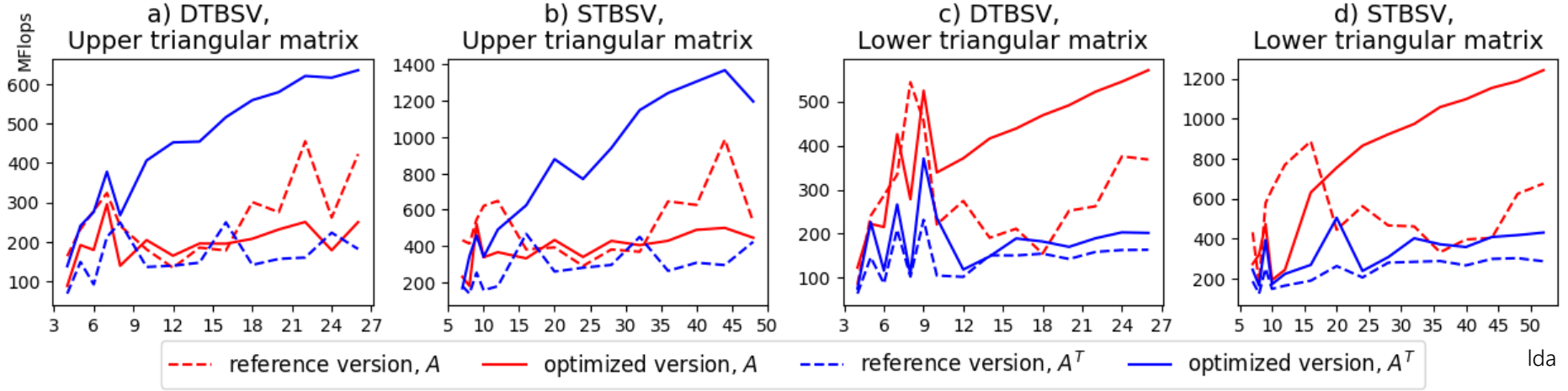
Вычислительные эксперименты. SBMV. Banana Pi BPI-F3, RVV 1.0



- Алгоритм 1 используется при малом числе диагоналей, алгоритм 2 – при **lda > 14** для double, **lda > 20** для float.
- Среднее ускорение – **1.6** раз

$n = 5$ млн.

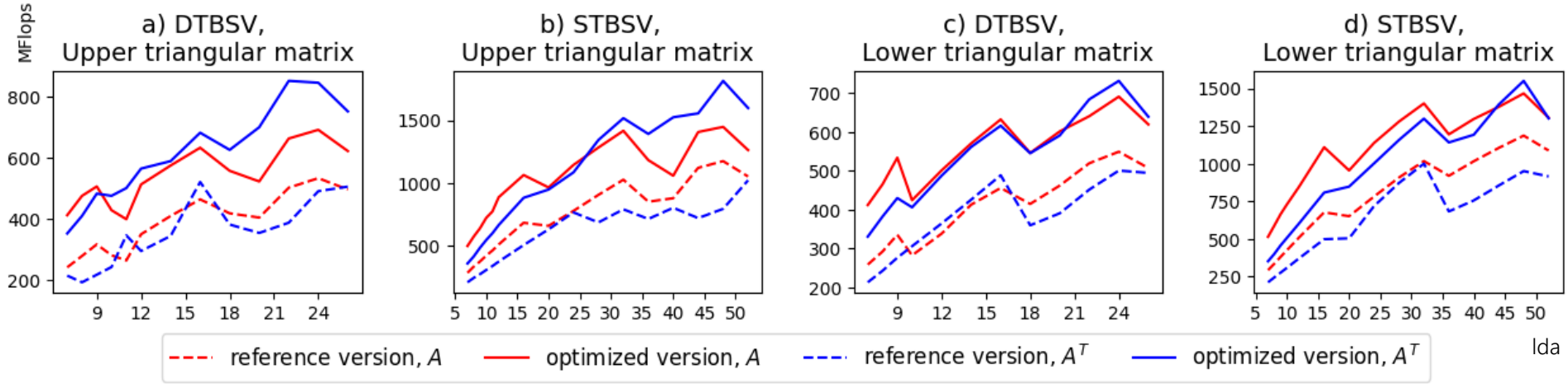
Вычислительные эксперименты. TBSV. Lichee Pi 4A, RVV 0.7.1



- 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**.

$n = 0.5$ млн.

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



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

$n = 0.5$ млн.

Заключение

- На основе реализации из библиотеки OpenBLAS предложены новые низкоуровневые реализации алгоритмов GBMV, TBMV, SBMV, TBSV для процессоров архитектуры RISC-V.
- Эффективность предложенных алгоритмов зависит от вида матрицы и числа ее диагоналей. Эмпирически были установлены критерии переключения между новыми реализациями и базовой версией.
- Финальные версии алгоритмов работают быстрее базовой реализации, до 7 раз быстрее на плате Lichee Pi 4A, до 4.9 раз быстрее на плате Banana Pi BPI-F3.

Спасибо за внимание!

Вопросы?