DOI: 10.15514/ISPRAS-2023-35(5)-7

О проблемах использования библиотеки OpenBLAS в продуктовом коде на RISC-V

К.А. Зайцева, ORCID: 0009-0009-8645-8795 <*k.zaytseva@yadro.com*>
В.В. Пузикова, ORCID: 0000-0003-0712-4519 <*v.puzikova@yadro.com*>
А.Д. Соколов, ORCID: 0009-0007-3404-5660 <*an.sokolov@yadro.com*>

ООО YADRO,
123022, Россия, г. Москва, ул. Рочдельская, д. 15, стр. 15.

Аннотация. Использование для численного решения задач механики сплошной среды метода граничных элементов приводит к необходимости решения системы линейных алгебраических уравнений с заполненной матрицей. Стандартами де-факто интерфейса программных реализаций функций над заполненными матрицами являются BLAS/LAPACK. Среди оптимизированных открытых реализаций BLAS/LAPACK, только библиотека OpenBLAS включает в себя оптимизации под самый широкий спектр аппаратных платформ – Intel, AMD, ARM и RISC-V. Экосистема открытой архитектуры RISC-V в настоящее время активно развивается: европейские суперкомпьютерные центры открыли центры компетенции RISC-V в рамках правительственной грантовой поддержки EuroHPC, поскольку решения, основанные на архитектуре ARM, не были признаны частью европейской инициативы по развитию собственной технологической независимости. В настоящее время в мире разрабатываются не только высокопроизводительные RISC-V процессоры, но и AI-ускорители, а также видеокарты на RISC-V архитектуре. OpenBLAS активно поддерживается и оптимизируется под появляющееся RISC-V оборудование и расширения. Однако, к библиотекам, использующимся в продуктовом коде, традиционно предъявляются серьезные требования по стабильности и надежности, чтобы минимизировать возможные ошибки и сбои в продукте. Как оказалось, с этой точки зрения, OpenBLAS имеет ряд проблем, которые нам пришлось решить с целью продуктизации этой библиотеки. В данной статье описывается тестовая система OpenBLAS, рассматриваются проблемы тестирования LAPACK-функционала библиотеки и пути их решения. Кроме того, анализируется тестовое покрытие BLAS-функционала и обсуждаются достигнутые результаты по его увеличению. В дальнейшем планируется внести описанные изменения в проект OpenBLAS.

Ключевые слова: метод граничных элементов; система линейных алгебраических уравнений с заполненной матрицей; OpenBLAS; LAPACK; RISC-V; тестирование; продуктизация.

Для цитирования: Зайцева К.А., Пузикова В.В., Соколов А.Д. О проблемах использования библиотеки OpenBLAS в продуктовом коде на RISC-V. Труды ИСП РАН, том 35, вып. 5, 2023 г., стр. 91–106. DOI: 10.15514/ISPRAS–2023–35(5)–7.

On Problems in OpenBLAS Library Usage in Productized Code on RISC-V

K.A. Zaytseva, ORCID: 0009-0009-8645-8795 <*k.zaytseva@ yadro.com*>
V.V. Puzikova, ORCID: 0000-0003-0712-4519 <*v.puzikova@yadro.com*>
A.D. Sokolov, ORCID: 0009-0007-3404-5660 <*an.sokolov@ yadro.com*>
YADRO,
15, bld.15, Rochdelskaya st., Moscow, 123022, Russia.

Abstract. The boundary element method usage for the numerical simulation in continuum mechanics problems leads to the need to solve a system of linear algebraic equations with a dense matrix. The de facto standards for the interface of functions over dense matrices and vectors software implementations are BLAS/LAPACK. Among the optimized open-source BLAS/LAPACK implementations, only the OpenBLAS library includes optimizations for the widest range of hardware platforms. This library is optimized for Intel, AMD, ARM and RISC-V architectures. The open RISC-V architecture ecosystem is currently actively developing. European supercomputing centers have opened RISC-V competence centers as part of the government's EuroHPC grant support, since solutions based on the ARM architecture are not recognized as part of the European initiative to develop its own technological independence. Currently, companies included in the international RISC-V consortium are developing not only high-performance RISC-V processors, but also AI accelerators, as well as video cards based on RISC-V architecture. OpenBLAS is actively supported and optimized for emerging RISC-V hardware and extensions. However, libraries used in product code are traditionally subject to strict requirements for stability and reliability in order to minimize possible errors and failures in the product. As it turned out, from this point of view, OpenBLAS has a number of problems that we had to solve in order to productize this library. In this article the OpenBLAS test system is described, the problems of testing the LAPACK functionality of the library and ways to solve them are discussed. In addition, the test coverage of the BLAS functionality is analyzed and the results achieved in increasing it are presented. It is planned to contribute the described changes to the OpenBLAS project.

Keywords: boundary element method; system of linear equations with dense matrix; OpenBLAS; LAPACK; RISC-V; testing; productization.

For citation: Zaytseva K.A., Puzikova V.V., Sokolov A.D. On Problems in OpenBLAS Library Usage in Productized Code on RISC-V. *Trudy ISP RAN/Proc. ISP RAS*, vol. 35, issue 5, 2023. pp. 91-106 (in Russian). DOI: 10.15514/ISPRAS-2023-35(5)-7.

1. Введение

Б*о*льшая часть задач механики сплошной среды не имеет аналитического решения и может быть решена только численно. Численные методы решения краевых задач механики сплошной среды глобально можно разделить на две группы.

Первую группу составляют методы, в которых происходит построение разностных аналогов исходных дифференциальных уравнений в частных производных во всей расчетной области. Методы этой группы можно разделить [1-2] на методы конечных разностей (МКР), методы контрольных объемов (МКО) и методы конечных элементов (МКЭ). МКР подразумевают замену производных из исходных уравнений на разностные на построенной в расчетной области сетке. В МКО ячейки построенной сетки являются контрольными объемами, в которых интегрируются уравнения, описывающие физические процессы в задаче. Затем, при помощи принципов сохранения массы, импульса и энергии, полученные уравнения связываются между соседними контрольными объемами через граничные условия. Наконец, в МКЭ, каждая ячейка сетки является конечным элементом, на котором выбирается вид аппроксимирующей функции, вне элемента равной нулю. Значения этих функций на границах элементов в узлах сетки являются решением задачи. Их коэффициенты определяются из условия равенства значений функций на границах между элементами [3].

Ко второй группе относят метод граничных элементов (МГЭ) или метод граничных интегральных уравнений [4-6]. Этот метод принципиально отличается от методов первой

группы тем, что сначала краевая задача механики сплошной среды, состоящая из дифференциальных уравнений в частных производных и краевых условий, при помощи формул Грина сводится к интегральному уравнению на границе расчетной области, а затем уже для этого граничного интегрального уравнения строится дискретный аналог. МГЭ был разработан в 1960-х годах и с тех пор претерпел множество изменений и усовершенствований. Сначала МГЭ использовался преимущественно для решения задач электростатики и потенциального течения. Он позволял точно моделировать эффекты на поверхности объекта, такие как распределение электрического заряда или давления. В дальнейшем МГЭ был расширен для решения уравнений упругости и теплопроводности, что позволило моделировать напряжения и деформации в материалах, а также тепловые потоки. В настоящее время в механике сплошной среды МГЭ находит широкое применение во многих областях, включая [4-5]:

- структурную механику – МГЭ используется для анализа напряжений и деформаций в конструкциях, таких как мосты, здания, автомобили и самолеты, что позволяет оценить прочность и надежность конструкций, а также оптимизировать их дизайн;
- геотехнику – МГЭ применяется для моделирования поведения грунтов и горных пород при различных нагрузках и условиях, анализа устойчивости склонов, расчета осадок при строительстве фундаментов и тоннелей, а также для моделирования поведения грунта при землетрясении;
- аэродинамику и гидродинамику – МГЭ применяется для моделирования потока воздуха или жидкости вокруг объектов, таких как крылья самолетов или корпуса судов, позволяет оценить силы сопротивления и подъемные силы, а также оптимизировать форму объекта для улучшения его аэродинамических или гидродинамических характеристик;
- акустику – МГЭ используется для моделирования звуковых полей и распространения звука в сплошных средах, анализа шумовых и вибрационных характеристик зданий, автомобилей или других объектов, а также для оптимизации звукового дизайна;
- электромагнетизм – МГЭ применяется для моделирования распределения электрического или магнитного поля вокруг объектов, анализа электромагнитной совместимости, проектирования антенн или оптимизации электромагнитных устройств.

Общей отличительной чертой всех численных методов первой группы (МКР, МКО, МКЭ) является разреженная структура матриц систем линейных алгебраических уравнений (СЛАУ), получающихся после дискретизации дифференциальных уравнений по времени и пространству. На всякий случай поясним, что разреженными называются матрицы, б*о*льшая часть элементов которых является нулями. В программной реализации для хранения таких матриц используются специальные форматы, такие как CSR, CSC, COO, ELL, JAD, BSR, BSC и так далее. [7], чтобы не расходовать память на хранение нулевых элементов. Для решения СЛАУ с разреженными матрицами, как правило, используются специальные итерационные методы, например, крыловские методы (CG, BiCGStab, FGMRES и др.), зачастую со специальными предобуславливателями, учитывающими разреженную структуру матрицы [8].

В результате использования метода граничных элементов [4-6], напротив, получается СЛАУ с заполненной матрицей, для решения которой используют прямые методы, а не перечисленные выше итерационные. Стандартом де-факто интерфейса программных реализаций функций линейной алгебры над заполненными матрицами и векторами является BLAS (Basic Linear Algebra Subprograms – базовые подпрограммы линейной алгебры). Этот стандарт был опубликован в 1979 г. [9] и использован, в частности, при создании LAPACK (Linear Algebra PACKage) – библиотеки для решения широкого спектра задач линейной

алгебры, таких как решение систем линейных уравнений, вычисление собственных значений и векторов, построение матричных разложений и мн. др. [10]. Таким образом, пакеты математических программ, позволяющие проводить моделирование методом граничных элементов, должны содержать либо собственные реализации отдельных необходимых функций BLAS/LAPACK, либо использовать сторонние BLAS/LAPACK библиотеки.

Так, например, один из наиболее универсальных инструментов для построения дискретных аналогов для граничных интегральных операторов и решения задач методом МГЭ, open-source проект BEM++ [11,12] может собираться с реализациями BLAS/LAPACK из библиотек Intel MKL, GotoBLAS и OpenBLAS [13]. В настоящее время использование первой из перечисленных библиотек может быть сопряжено с юридическими рисками, а проект GotoBLAS с 2008 года не обновляется. Поэтому наиболее предпочтительным является использование для BEM++ библиотеки OpenBLAS.

Необходимо отметить, что среди других оптимизированных высокопроизводительных открытых реализаций BLAS/LAPACK, таких как, например, Eigen [14] и Armadillo [15], библиотека OpenBLAS выделяется оптимизациями под самый широкий спектр аппаратных платформ – Intel, AMD, ARM и RISC-V, причем под RISC-V [16] к настоящему моменту оптимизирована только она одна. В то же время экосистема открытой архитектуры RISC-V сейчас начинает активно развиваться в направлении HPC (High Performance Computing): ведущие европейские HPC центры, такие как BSC (Barcelona Supercomputing Center) и EPCC (Edinburgh Parallel Computing Centre), открыли центры компетенции RISC-V в рамках правительственной грантовой поддержки EuroHPC [17]. Это вызвано тем, что решения, основанные на архитектуре ARM, не были признаны частью европейской инициативы по развитию собственной технологической независимости. Первые RISC-V HPC системы ожидаются в 2025-2026 гг. в рамках проекта европейского суперкомпьютера BSC MareNostrum 6. Идет разработка не только высокопроизводительных RISC-V CPU, таких как 432-ядерный Occamy [18] от ETH или Atrevido 423 от Semidynamics, но и AI-ускорителей в рамках проекта EUPILOT [19]. Кроме того, существует проект разработки RISC-V GPU Vortex: сейчас с ним можно работать на FPGA, причем имеется конвейер для запуска программ, написанных на CUDA.

Целью данной работы является исследование стабильности и надежности работы библиотеки OpenBLAS на архитектуре RISC-V, а также минимизация возможных ошибок и сбоев в функционале OpenBLAS, которые ухудшают качество продуктов, использующих OpenBLAS на RISC-V. Это актуально, поскольку в случае использования OpenBLAS на RISC-V имеется сразу два потенциальных источника проблем. Во-первых, OpenBLAS как и многие другие open-source проекты может изначально иметь проблемы с качеством тестирования. Во-вторых, библиотека OpenBLAS оперативно оптимизируется большим сообществом под появляющееся RISC-V оборудование и расширения RISC-V ISA (Instruction Set Architecture), а портирование библиотек на новые архитектуры зачастую может приводить к некорректной работе функционала. Если начать портировать на RISC-V другие открытые HPC библиотеки, использующие OpenBLAS, такие как упомянутый выше BEM++, не убедившись в корректности и стабильности работы OpenBLAS на RISC-V, процесс отладки сильно осложнится.

2. Библиотека OpenBLAS

Библиотека OpenBLAS (Open Basic Linear Algebra Subprograms) является открытым проектом, разработанным для эффективного выполнения операций линейной алгебры на многоядерных процессорах. Она предоставляет набор оптимизированных операций над заполненными матрицами и векторами, таких как умножение матриц, решение систем линейных уравнений и вычисление собственных значений. В библиотеке реализованы BLAS и LAPACK API.

2.1 История развития

История создания и развития OpenBLAS началась в 2002 году [13] с проекта GotoBLAS, разработанного Кадзусигэ Гото (Kazushige Goto) и Робертом ван де Гейном (Robert van de Geijn) в Университете Техаса в Остине. GotoBLAS представляла собой оптимизированную библиотеку базовых линейных алгебраических подпрограмм, в которой использовались ассемблерные инструкции для достижения высокой производительности.

В 2008 году проект GotoBLAS был переименован в ATLAS (Automatically Tuned Linear Algebra Software) и стал открытым проектом с открытым исходным кодом. Библиотека ATLAS предоставляла возможность автоматической настройки для конкретного аппаратного обеспечения, что позволяло достичь еще большей производительности.

В 2011 году разработчики ATLAS объединили свои усилия с коллективом проекта LAPACK (Linear Algebra PACKage) и создали новый проект под названием OpenBLAS. OpenBLAS была разработана с использованием оптимизаций из ATLAS и предоставляла полную совместимость с LAPACK. С тех пор OpenBLAS продолжает развиваться и улучшаться благодаря активному вкладу большого сообщества разработчиков.

2.2 Функционал OpenBLAS

OpenBLAS полностью реализует интерфейсы BLAS и LAPACK, а также предоставляет дополнительное множество «BLAS-like» функций для расширения функциональности BLAS. BLAS-часть библиотеки содержит в себе набор базовых операций линейной алгебры, чаще всего использующихся в прикладных программах. Все функции оптимизированы, как алгоритмически, так и на микроархитектурном уровне, и на их основе строятся реализации более сложных алгоритмов, входящих в LAPACK, PBLAS (Parallel Basic Linear Algebra Subprograms), ScaLAPACK(Scalable Linear Algebra PACKage).

Функции BLAS делятся на следующие 3 группы, называемые «уровнями»:

- 1 уровень – операции над векторами (например, скалярное произведение или умножение вектора на скаляр);
- 2 уровень – векторно-матричные операции (например, умножение матрицы на вектор, причем вид матрицы учитывается при оптимизации алгоритма);
- 3 уровень – матрично-матричные операции (например, умножение матрицы на матрицу или решение СЛАУ с треугольной матрицей).

Интерфейсы BLAS для языков Fortran (BLAS) и C (CBLAS) отличаются только списком аргументов и способом их обработки. В основе каждой функции лежит один и тот же код, написанный на языке C с использованием низкоуровневых оптимизаций и поддержкой многопоточности. Например, для умножения вектора на скаляр и сложения результата с вектором в случае Fortran вызывается функция `saxpy()`, а в случае C/C++ – `cblas_saxpy()`.

LAPACK-часть библиотеки можно разделить на четыре большие группы функций, каждая из которых включает в себя также и выполнение ряда сопутствующих вычислительных задач:

1) построение разложений плотных матриц, таких как разложения LU, QR, Холецкого, Шура, SVD и др.;
2) решение систем линейных алгебраических уравнений с плотными матрицами различных видов (симметричными, несимметричными, эрмитовыми и т.д.);
3) решение линейных задач наименьших квадратов;
4) решение задач на собственные значения.

Каждая группа содержит не только вычислительные функции, но и функции-драйверы, обеспечивающие решение общих задач. Вычислительные функции вызываются из функций-драйверов и выполняют различные подзадачи, например, LU-разложение для функции-драйвера решения СЛАУ.

Также как и BLAS часть LAPACK имеет интерфейс для использования в программах, написанных на языке C/C++. Этот интерфейс называется LAPACKE. Он представляет собой функции-обертки для LAPACK функций. Малая часть LAPACK функциональности реализована непосредственно в OpenBLAS на языке C и содержит дополнительные оптимизации для исполнения в многопоточной среде. Для предоставления полного множества функций LAPACK и LAPACKE в библиотеке OpenBLAS используется реализация из групп библиотек Netlib, написанная на языке Fortran. Кроме того, на случай отсутствия Fortran компилятора в OpenBLAS предусмотрен вариант сборки с использованием исходных файлов реализации LAPACK из Netlib, сконвертированных на язык C с помощью f2clapack скрипта из репозитория PRIMME [20].

2.3 Система сборки OpenBLAS

Библиотека OpenBLAS имеет развитую систему сборки с множеством ключей, позволяющих осуществлять:

- выбор целевой архитектуры сборки с микроархитектурными оптимизациями;
- включение динамической диспетчеризации оптимизаций, когда собранная библиотека содержит оптимизации для нескольких микроархитектур и может определять оптимальную оптимизацию в ходе исполнения программы;
- выбор между использованием пары компиляторов C/Fortran и использованием только одного C компилятора;
- включение/отключение многопоточного режима с использованием технологий параллельного программирования PThreads (POSIX Threads) и OpenMP (Open Multi-Processing);
- выбор ширины знакового целочисленного типа в интерфейсах функций (32/64-битные).

Рассмотрим подробнее выбор целевой архитектуры. Как уже отмечалось выше, OpenBLAS поддерживает многие архитектуры: X86/X86_64, PowerPC, MIPS, ARM, Elbrus, RISC-V и др. Для каждой из перечисленных архитектур в библиотеке OpenBLAS имеются оптимизации для конкретной микроархитектуры, учитывающие особенности набора команд и размеров кэшей. Для рассматриваемой в данной статье архитектуры RISC-V в OpenBLAS в настоящее время существует три варианта сборки:

1) Generic – включает только скалярные инструкции без векторных оптимизаций;
2) RVV 0.7.1 – включает низкоуровневые оптимизации с использованием векторных инструкций из RISC-V Vector Extension v.0.7.1;
3) RVV v.1.0 – включает низкоуровневые оптимизации с использованием векторных инструкций из RISC-V Vector Extension v.1.0.

Здесь необходимо пояснить, что последней ратифицированной версий векторного расширения RISC-V является RVV 1.0, однако в настоящее время RISC-V процессоры с поддержкой данного расширения все еще недоступны в свободной продаже. Промежуточная версия векторного расширения, RVV 0.7.1, поддерживается в ядрах C906 и C910 компании XuanTie. Эти ядра входят в состав процессоров на доступных для покупки платах Sipeed Nezha, Mango Pi, Lichee Pi 4A и др. По этим причинам далее мы будем рассматривать только Generic и RVV 0.7.1 сборки. Отметим, что многопоточные сборки библиотеки не будут рассматриваться в данной статье.

2.4 Тестовая система OpenBLAS

Существует пять открытых наборов тестов для тестирования частей функционала OpenBLAS:

1) тесты для BLAS API, заимствованные из Netlib BLAS с минимальными изменениями (написаны на языке Fortran);
2) тесты для CBLAS API, аналогичные BLAS API тестам, но адаптированные для тестирования CBLAS (изначально написаны на Fortran, но есть возможность сборки с вариантом тестов, сконвертированных на C);
3) собственный OpenBLAS фреймворк Utest для тестирования, схожий с фреймворком GTests (Google Test) и содержащий специфические сценарии тестирования, например, воспроизводители ошибок или вырожденных случаев;
4) отдельный проект BLAS-Tester [21] для тестирования BLAS алгоритмов и сравнения производительности со сторонними реализациями BLAS;
5) тесты для LAPACK API, заимствованные из Netlib BLAS с минимальными изменениями (написаны на языке Fortran).

Отметим, что проект BLAS-Tester является более гибким в настройке, чем исходные тесты из Netlib. Так, например, в тестах из Netlib размеры матриц/векторов указываются по отдельности, в то время как BLAS-Tester позволяет задать сразу диапазон значений с определённым шагом. Кроме того, в BLAS-Tester имеется возможность указывать дополнительные параметры для тестов, например, смещения адресов массивов для тестирования с невыровненной памятью.

Также требует пояснений пятая группа тестов – тестов LAPACK, поскольку это отдельная большая система для тестирования более чем тысячи функций. Тесты этой группы делятся на 18 подгрупп для функций над аргументами одинарной точности и на 19 подгрупп для функций над аргументами двойной точности. Разделение на подгруппы производится не только по типам решаемых задач, но и по способам хранения матриц. Каждая подгруппа тестов запускается с помощью собственного исполняемого файла, которому однозначно соответствует входной файл с параметрами тестирования. Этот файл содержит не только размеры и типы тестовых матриц, но и значения параметров исполнения тестируемых функций, передаваемые в качестве переменных окружения выполнения, а не аргументов. Результатом выполнения тестовой программы служит текстовый файл с логом запуска тестов и подробным описанием возникших ошибок в случае их наличия.

Подчеркнем, что тестирование функций на точность не использует каких бы то ни было заданных эталонных результатов, а опирается исключительно на математические свойства тестируемых алгоритмов (нормы невязки и т.д.). Например, для функции, реализующей LU-разложение матрицы A, значение ошибки δ вычисляется по формуле

$$\delta = \frac{|\mathrm{L} * \mathrm{U} - \mathrm{A}|_1}{\mathrm{N} * |\mathrm{A}|_1 * \varepsilon}, \quad (1)$$

где ε – относительная машинная точность, $|\cdot|_1$ – первая матричная норма (максимальная сумма элементов столбца), N – размер квадратной матрицы. Соответственно, тест будет считаться пройденным, если значение δ меньше заданного при запуске тестов порога.

3. Тестирование библиотеки OpenBLAS на RISC-V

Для сборки библиотеки использовался GCC (GNU Compiler Collection) тулчейн с открытым исходным кодом от компании T-HEAD [22]. Он включает в себя компиляторы для языка C (gcc с поддержкой набора инструкций RVV 0.7.1) и Fortran (gfortran), а также симулятор Qemu с поддержкой RVV 0.7.1. Запуски и отладка осуществлялись не только на Qemu, но и на плате Mango Pi с SoC (System On Chip) AllWinner D1, включающим одно ядро C906 компании XuanTie с поддержкой RVV 0.7.1.

3.1 Анализ тестового покрытия BLAS-функционала и его увеличение

Тесты – важная часть качественного и надежного программного продукта. При использовании открытых библиотек в своих приложениях важно оценивать покрытие кода тестами и при необходимости увеличивать его.

Для того чтобы проанализировать покрытие BLAS-функционала библиотеки OpenBLAS тестами использовалась утилита LCOV [23]. Чтобы собрать отчет о тестовом покрытии, необходимо построить библиотеку с включенной инструментацией. Для этого необходимо добавить ключи сборки `-fprofile-arcs` и `-ftest-coverage`. В результате после построения библиотеки также должны сгенерироваться файлы с расширением gcno.

Теперь нужно создать «базовый уровень» LCOV – файл данных покрытия, содержащих нулевое покрытие для каждой инструментированной строки проекта. Это требуется для того, чтобы получить данные о покрытии всех файлов исходного кода, участвующих в сборке, даже тех, которые не будут использоваться непосредственно во время запуска тестов. Если OpenBLAS компилировался при помощи gcc, то «базовый уровень» LCOV создается при помощи следующей команды:

```
lcov --gcov-tool  <PATH_TO_TOOLCHAIN>/bin/riscv64-unknown-linux-
gnu-gcov --capture --initial --directory driver/ --directory
kernel/--directory interface/ --output-file coverage_base.info
```

Если же для компиляции библиотеки OpenBLAS использовался компилятор clang, возникает проблема с некорректной обработкой пробела между `llvm-cov` и `gcov`. Для решения этой проблемы удобно использовать небольшой скрипт, llvm-gcov.sh, см. листинг 1.

```
#!/bin/bash
exec <path-to-clang-toolchain>/llvm/bin/llvm-cov gcov "$@"
```

Листинг 1. Скрипт llvm-gcov.sh.
Listing 1. Script llvm-gcov.sh.

Данный скрипт необходимо сделать исполняемым и использовать как `gcov-tool` при создании «базового уровня» LCOV:

```
lcov --gcov-tool llvm-gcov.sh --capture –initial --directory driver/
--directory kernel/ --directory interface/ --output-file
coverage_base.info
```

После этого можно запускать BLAS-тесты. Отметим, что помимо тестов из OpenBLAS в исследовании также использовались тесты из проекта BLAS-Tester. По мере прохождения тестов генерируются файлы с расширением gcda. После прохождения всех тестов необходимо снова запустить LCOV, в случае использования gcc при помощи команды

```
lcov --gcov-tool  <PATH_TO_TOOLCHAIN>/bin/riscv64-unknown-linux-
gnu-gcov --capture --directory driver/ --directory kernel/ --
directory interface/ --output-file coverage_tests.info
```

а в случае clang

```
lcov --gcov-tool llvm-gcov.sh --capture --directory driver/ --
directory kernel/ --directory interface/ --output-file
coverage_tests.info
```

Теперь можно объединить файлы трассировки:

```
lcov --add-tracefile coverage_base.info --add-tracefile
     coverage_tests.info --output-file coverage_total.info
```

Перед генерацией отчета о тестовом покрытии необходимо удалить ненужные исходные файлы из файлов трассировки:

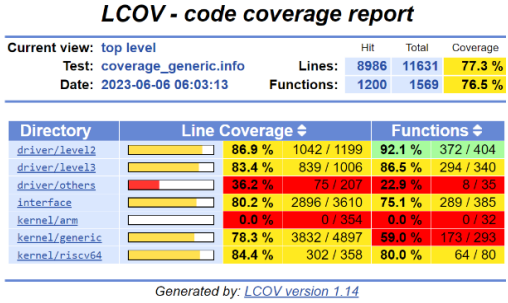
```
lcov --remove coverage_total.info -o coverage_total_filtered.info
<path_to_openblas>/interface/lapack/*
<path_to_riscv>/sysroot/usr/include/* <path_to_openblas>/common*
<path_to_openblas>/symcopy.h
```

После выполнения всех этих действий можно сгенерировать отчет о покрытии кода библиотеки тестами, выполнив следующую команду:

```
genhtml --output-directory html coverage_total_filtered.info
```

В итоге в html директории должен появиться файл index.html с отчетом. Отчеты LCOV для Generic (рис. 1) и RVV 0.7.1 (рис. 2) сборок OpenBLAS показали, что покрытие кода тестами не превышает 79%.

Для функций с покрытием менее 60% были написаны недостающие тесты и включены в наш форк OpenBLAS, после чего отчеты были собраны снова. Результаты представлены на рис. 3 и 4.



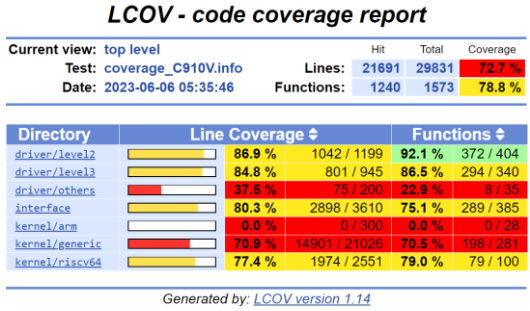
LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level				
Test:	coverage_generic.info	Lines:	8986	11631	77.3 %
Date:	2023-06-06 06:03:13	Functions:	1200	1569	76.5 %

Directory	Line Coverage		Functions	
driver/level2	86.9 %	1042 / 1199	92.1 %	372 / 404
driver/level3	83.4 %	839 / 1006	86.5 %	294 / 340
driver/others	36.2 %	75 / 207	22.9 %	8 / 35
interface	80.2 %	2896 / 3610	75.1 %	289 / 385
kernel/arm	0.0 %	0 / 354	0.0 %	0 / 32
kernel/generic	78.3 %	3832 / 4897	59.0 %	173 / 293
kernel/riscv64	84.4 %	302 / 358	80.0 %	64 / 80

Generated by: LCOV version 1.14

Рис. 1. Отчет LCOV о тестовом покрытии Generic сборки OpenBLAS.
Fig. 1. LCOV report on test coverage of OpenBLAS Generic build.



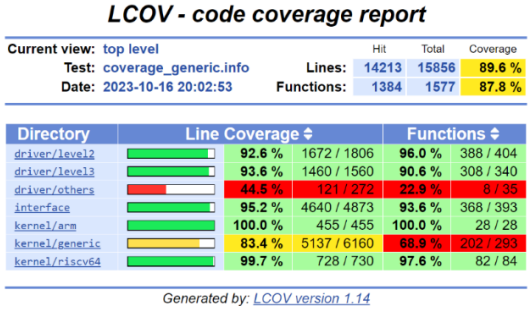
LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level				
Test:	coverage_C910V.info	Lines:	21691	29831	72.7 %
Date:	2023-06-06 05:35:46	Functions:	1240	1573	78.8 %

Directory	Line Coverage		Functions	
driver/level2	86.9 %	1042 / 1199	92.1 %	372 / 404
driver/level3	84.8 %	801 / 945	86.5 %	294 / 340
driver/others	37.5 %	75 / 200	22.9 %	8 / 35
interface	80.3 %	2898 / 3610	75.1 %	289 / 385
kernel/arm	0.0 %	0 / 300	0.0 %	0 / 28
kernel/generic	70.9 %	14901 / 21026	70.5 %	198 / 281
kernel/riscv64	77.4 %	1974 / 2551	79.0 %	79 / 100

Generated by: LCOV version 1.14

Рис. 2. Отчет LCOV о тестовом покрытии RVV 0.7.1 сборки OpenBLAS.
Fig. 2. LCOV report on test coverage of OpenBLAS RVV 0.7.1 build.



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level				
Test:	coverage_generic.info	Lines:	14213	15856	89.6 %
Date:	2023-10-16 20:02:53	Functions:	1384	1577	87.8 %

Directory	Line Coverage		Functions	
driver/level2	92.6 %	1672 / 1806	96.0 %	388 / 404
driver/level3	93.6 %	1460 / 1560	90.6 %	308 / 340
driver/others	44.5 %	121 / 272	22.9 %	8 / 35
interface	95.2 %	4640 / 4873	93.6 %	368 / 393
kernel/arm	100.0 %	455 / 455	100.0 %	28 / 28
kernel/generic	83.4 %	5137 / 6160	68.9 %	202 / 293
kernel/riscv64	99.7 %	728 / 730	97.6 %	82 / 84

Generated by: LCOV version 1.14

Рис. 3. Отчет LCOV о тестовом покрытии Generic сборки OpenBLAS после проведения работ по его увеличению и исправлению обнаруженных ошибок.
Fig. 3. LCOV report on test coverage of OpenBLAS Generic build after work to increase it and fix detected bugs.

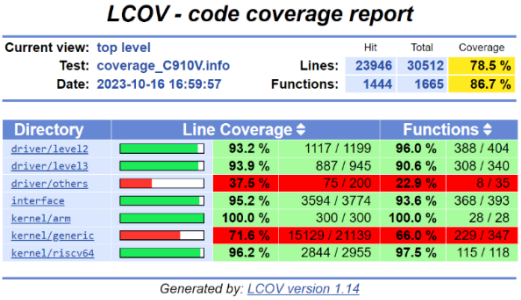


Рис. 4. Отчет LCOV о тестовом покрытии RVV 0.7.1 сборки OpenBLAS после проведения работ по его увеличению и исправлению обнаруженных ошибок.
Fig. 4. LCOV report on test coverage of OpenBLAS RVV 0.7.1 build after work to increase it and fix detected bugs.

В результате суммарное покрытие BLAS функций второго и третьего уровня превысило 92%, а покрытие функций с RISC-V оптимизациями – 96%. При этом добавленные тесты позволили выявить и устранить ряд ошибок в BLAS-функционале, например:

- `imatcopy()` – ошибка выделения памяти и некорректная обработка краевых случаев для комплексных матриц, затрагивающая все архитектуры;
- `gemmt()` – некорректная обработка комплексно-сопряженных матриц, затрагивающая все архитектуры;
- `dsdot()` – отсутствие реализации для RVV 0.7.1 (использовалась реализация `sdot()`, не обеспечивавшая необходимой точности);
- `asum()` и `axpby()` – некорректная работа оптимизаций при некоторых значениях инкремента;
- `i[s/d]max/min()` – ошибка в приведении типов.3.2 Решение проблем с прохождением тестов LAPACK-функционала

Перейдем к тестированию LAPACK-функционала библиотеки OpenBLAS. Для этого запустим описанный выше набор LAPACK тестов для Generic и RVV 0.7.1 сборок библиотеки с помощью симулятора Qemu, а также на плате Mango Pi. Для каждого набора оптимизаций будет протестировано четыре конфигурации библиотеки:

1) «Fortran, Int32» – сборка с использованием исходных файлов LAPACK на Fortran и 32-битными целочисленными типами;
2) «Fortran, Int64» – сборка с использованием исходных файлов LAPACK на Fortran и 64-битными целочисленными типами;
3) «C, Int32» – сборка с использованием файлов LAPACK, сконвертированных на C, и 32-битными целочисленными типами;
4) «C, Int64» – сборка с использованием файлов LAPACK, сконвертированных на C, и 64-битными целочисленными типами;

Общее количество LAPACK-тестов для каждого типа данных представлено в табл. 1. Однако, первая попытка запуска тестов для Generic сборки OpenBLAS показала, что по непонятным причинам запускаются не все тесты (табл. 2). Кроме того, в конфигурации «C, Int64» падает в сумме по всем типам данных около 9500 тестов (табл. 3).

Табл. 1 Количество запускаемых LAPACK тестов.
Table 1. Number of launched LAPACK tests.

Тип данных	Float	Double	Complex Float	Complex Double
Число тестов	1 327 023	1 327 845	786 775	787 842

Табл. 2 Количество пропущенных LAPACK тестов для Generic сборки
Table 2. Number of skipped LAPACK tests for Generic build

Конфигурация	Float	Double	Complex Float	Complex Double
Fortran, Int32	9 336 (0.70%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Fortran, Int64				
C, Int32	10 260 (0.77%)		385 (0.00%)	
C, Int64	529 067 (39.86%)	518 817 (39.07%)	23 907 (3.03%)	23 778 (3.01%)

Для RVV 0.7.1 сборки результаты запуска тестов еще хуже (табл. 4 и 5): около 75% тестов не запускается, все тесты при использовании сконвертированных файлов для double зависают. Анализ результатов тестирования RVV 0.7.1 сборки показал, что большинство тестов пропущены по причине ошибок сегментации, возникающих из-за выхода за пределы массива. Кроме того, были зафиксированы ошибки в вычислениях и зависания тестов.

Табл. 3 Статистика падений LAPACK тестов для Generic сборки
Table 3. Fail statistics of LAPACK tests for Generic build

Конфигурация	Float	Double	Complex Float	Complex Double
Fortran, Int32	3 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Fortran, Int64				
C, Int32	1 (0.00%)		1 (0.00%)	
C, Int64	2 206 (0.27%)	2 220 (0.27%)	2 469 (0.32%)	2 469 (0.32%)

Табл. 4 Количество пропущенных LAPACK тестов для RVV 0.7.1 сборки
Table 4. Number of skipped LAPACK tests for RVV 0.7.1 build

Конфигурация	Float	Double	Complex Float	Complex Double
Fortran, Int32	584 250 (44.02%)	1 297 545 (97.71%)	507 395 (64.49%)	527 820 (66.99%)
Fortran, Int64				
C, Int32	583 158 (43.94%)	Нет данных (зависания)	513 746 (65.29%)	534 042 (67.78%)
C, Int64	1 101 965 (83.04%)		531 302 (67.52%)	551 598 (70.01%)

Табл. 5 Статистика падений LAPACK тестов для RVV 0.7.1 сборки
Table 5. Fail statistics of LAPACK tests for RVV 0.7.1 build

Конфигурация	Float	Double	Complex Float	Complex Double
Fortran, Int32	404 (0.05%)	4 033 (13.31%)	2 076 (0.74%)	0 (0.00%)
Fortran, Int64				
C, Int32	382 (0.05%)	Нет данных (зависания)	2 077 (0.74%)	
C, Int64	2 620 (1.16%)		4 549 (1.78%)	2 471 (1.00%)

Поиск источников перечисленных проблем было решено начинать с Generic сборки, поскольку возникло предположений, что Generic реализация BLAS-функций, вызываемых из LAPACK алгоритмов, корректна и ошибки связаны с использованием сконвертированных на язык C файлов. Отладка производилась с помощью пары Qemu и GDB, которая позволяет отлаживать программы без использования конкретного RISC-V процессора. В результате было установлено, что сконвертированные на язык C файлы LAPACK из репозитория OpenBLAS содержат две проблемы, связанные с их взаимодействием с тестовой системой.

Первая проблема заключается в том, что размер типа logical, заменяющего на C тип LOGICAL из Fortran, в C файлах равен 32 битам. Однако при сборке Fortran кода используется ключ компиляции `-fdefault-integer-8`, который изменяет ширину не только INTEGER типа, но и LOGICAL. Отсутствие ошибок компиляции обусловлено особенностью передачи явных аргументов в Fortran-функции исключительно по указателю. После исправления этой проблемы количество запущенных тестов стало соответствовать другим конфигурациям, однако остались падения тестов с неверными статусами.

Эти падения были связаны со второй проблемой. Она заключалась в ошибке конвертации файлов, в которых функция обработки ошибок `xerbla()`, реализованная в тестовой системе на Fortran, вызывается с неверным количеством аргументов. Дело в том, что для каждой функции на Fortran неявно указываются последним аргументом размеры строк, переданные в эту функцию (листинг 2). Исправления этой проблемы представлены в листинге 3.

После исправления этих двух проблем число падений тестов для Generic сборки стало незначительным (табл. 6 и 7). Эти падения связаны исключительно с проблемами в точности алгоритмов для некоторых тестовых матриц. С пропусками менее 0.8% тестов float функций для всех конфигураций еще предстоит разобраться.

```
// Реализация на Fortran
SUBROUTINE XERBLA(SRNAME, INFO)
// Вызов функции на C
xerbla_("CGBCON", &i__1);
```

Листинг 2. Вызов xerbla_ из Си кода.
Listing 2. xerbla_ call from C code.

```
// Вызов функции на C
xerbla_("CGBCON", &i__1, (ftnlen)6);
```

Листинг 3. Исправленный вызов xerbla_ из Си кода.
Listing 3. Corrected xerbla_ call from C code.

Табл. 6 Количество пропущенных LAPACK тестов для Generic сборки после исправлений
Table 6. Number of skipped LAPACK tests for Generic build after fixes

Конфигурация	Float	Double	Complex Float	Complex Double
Fortran, Int32	9 336 (0.70%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Fortran, Int64				
C, Int32	10 260 (0.77%)		385 (0.00%)	
C, Int64				

Табл. 7 Статистика падений LAPACK тестов для Generic сборки после исправлений
Table 7. Fail statistics of LAPACK tests for Generic build after fixes

Конфигурация	Float	Double	Complex Float	Complex Double
Fortran, Int32	3 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)
Fortran, Int64				
C, Int32	1 (0.00%)		1 (0.00%)	
C, Int64				

Следующим шагом было исправление падений и зависаний LAPACK тестов для RVV 0.7.1 сборки библиотеки. Оказалось, что они связаны с проблемами в реализации вызываемых BLAS-функций с векторными оптимизациями под RISC-V. В ходе отладки были обнаружены и исправлены следующие проблемы в функциях:

- `idamax()` – ошибка в использовании RVV интринсиков, приводящая к индексации по значению переменной беззнакового типа, которой могло быть присвоено

отрицательное значение (в результате чего возникал SegFault);

- `dgemm()` – ошибка в использовании на ассемблере неверной инструкции для обнуления float регистра (в результате старшие 32 бита остались ненулевыми и могли появляться NaN значения);
- `nrm2()` – ошибка при обработке последней части массива, так называемого «хвоста», длина которого меньше длины векторного регистра (в результате чего функция возвращала неверный результат).

Таким образом, несмотря на имеющуюся развитую систему тестирования BLAS, некоторые проблемы в BLAS функционале оказалось возможным определить только благодаря LAPACK тестам.

Финальная статистика прохождения LAPACK-тестов для RVV 0.7.1 сборки представлена в табл. 8 и 9. В результате проведенной работы количество запущенных тестов стало почти соответствовать ожидаемому, однако с некоторыми тестами для комплексных чисел float точности еще предстоит разобраться.

Табл. 8 Количество пропущенных LAPACK тестов для RVV 0.7.1 сборки после исправлений
Table 8. Number of skipped LAPACK tests for RVV 0.7.1 build after fixes

<table>
<tr><th>Конфигурация</th><th>Float</th><th>Double</th><th>Complex Float</th><th>Complex Double</th></tr>
<tr><td>Fortran, Int32</td><td rowspan="2">1092 (0.00%)</td><td rowspan="4">0 (0.00%)</td><td rowspan="2">0 (0.00%)</td><td rowspan="4">0 (0.00%)</td></tr>
<tr><td>Fortran, Int64</td></tr>
<tr><td>C, Int32</td><td rowspan="2">0 (0.00%)</td><td rowspan="2">895 (0.11%)</td></tr>
<tr><td>C, Int64</td></tr>
</table>

Табл. 9 Статистика падений LAPACK тестов для RVV 0.7.1 сборки после исправлений
Table 9. Fail statistics of LAPACK tests for RVV 0.7.1 build after fixes

<table>
<tr><th>Конфигурация</th><th>Float</th><th>Double</th><th>Complex Float</th><th>Complex Double</th></tr>
<tr><td>Fortran, Int32</td><td rowspan="2">5 (0.00%)</td><td rowspan="4">0 (0.00%)</td><td rowspan="2">0 (0.00%)</td><td rowspan="4">0 (0.00%)</td></tr>
<tr><td>Fortran, Int64</td></tr>
<tr><td>C, Int32</td><td rowspan="2">0 (0.00%)</td><td rowspan="2">2 (0.00%)</td></tr>
<tr><td>C, Int64</td></tr>
</table>

4. Заключение

Несмотря на большое сообщество разработчиков и оперативную поддержку новых архитектур, библиотека OpenBLAS имеет ряд проблем, связанных с невысоким тестовым покрытием BLAS-функционала и включенными в состав библиотеки, очевидно без должного тестирования, сконвертированными на язык C исходными файлами с LAPACK функционалом из библиотек группы Netlib. Эти проблемы были выявлены при анализе Generic и RVV 0.7.1 сборок OpenBLAS.

Тестовое покрытие BLAS-функционала, собранное при помощи утилиты LCOV, не превышало 79% для обеих сборок. С целью продуктизации библиотеки были покрыты тестами функции с начальным покрытием менее 60%, что позволило повысить суммарное покрытие BLAS функций второго и третьего уровня до 92% и выше, а покрытие функций с RISC-V оптимизациями – до 96-99%. Благодаря добавленным тестам были обнаружены и устранены несколько ошибок в BLAS-функционале библиотек OpenBLAS.

При попытке запуска LAPACK тестов было обнаружено, что значительная часть тестов не запускается в случае сборки OpenBLAS на основе сконвертированных файлов с кодом LAPACK функций. Были локализованы ошибки, содержащиеся в сконвертированных файлах

из репозитория OpenBLAS. Их устранение позволило исправить ситуацию с запуском LAPACK тестов для Generic сборки. Отладка RVV 0.7.1 сборки показала, что падения и зависания тестов LAPACK функционала связаны с ошибками в реализации векторных оптимизаций в BLAS функциях `idamax()`, `dgemm()` и `nrm2()`, которые не обнаруживались без LAPACK тестов, несмотря на развитую систему тестирования BLAS-функционала.

В дальнейшем планируется внести полученные изменения (новые тесты и исправления ошибок) в проект OpenBLAS. Кроме того, становится возможным перейти к портированию на RISC-V открытых HPC библиотек, использующих OpenBLAS, таких как, например, библиотека BEM++ для моделирования методом граничных элементов.

Список литературы / References

[1]. Charpa S., Canale R. Numerical Methods for Engineers. McGraw Hill, 2014. 992 p.
[2]. Moin P. Fundamentals of Engineering Numerical Analysis. Cambridge University Press, 2010. 256 p.
[3]. Zienkiewicz O. C., Taylor R. L. The Finite Element Method: Its Basis and Fundamentals. Butterworth-Heinemann, 2013. 756 p.
[4]. Katsikadelis J. T. The Boundary Element Method for Engineers and Scientists, Second Edition: Theory and Applications. Academic Press, 2016. 464 p.
[5]. Brebbia C. A., Walker S. Boundary Element Techniques in Engineering. Newnes, 2016. 210 p.
[6]. Gwinner J., Stephan E. P. Advanced Boundary Element Methods: Treatment of Boundary Value, Transmission and Contact Problems. Springer, 2018. 670 p.
[7]. Garney S., Heroux M. A., Li G., Pozo R., Remington K. A., Wu K. A Revised Proposal for a Sparse BLAS Toolkit, Available at: https://sdm.lbl.gov/~kewu/ps/SparseBLAS96.pdf, accessed 23.10.2023.
[8]. Saad Y., Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, 2003. 184 p.
[9]. Lawson C. L., Hanson R. J., Kincaid D. R., Krogh F. Basic linear algebra subprograms for FORTRAN usage. ACM Transactions on Mathematical Software, vol. 5, issue 3, 1979, pp. 308-323. DOI: 10.1145/355841.355847.
[10]. Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Croz J. D., Greenbaum A., Hammarling S., McKenney A., Sorensen D. LAPACK Users' Guide. Society for Industrial and Applied Mathematics, 1987. 429 p.
[11]. BEM++, Available at: https://github.com/bempp/bempp-legacy/tree/master, accessed 23.10.2023.
[12]. Лукашин П.С., Стрижак С.В., Щеглов Г.А. Тестирование возможностей открытого кода BEM++ по решению задач акустики. Труды ИСП РАН, том 29, вып. 1, 2017, стр. 39-52. / Lukashin P.S., Strijhak S.V., Shcheglov G.A. Validation of open-source BEM++ code for simulation of acoustics problems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 1, 2017, pp. 39-52 (in Russian).
[13]. OpenBLAS, Available at: https://github.com/OpenMathLib/OpenBLAS, accessed 23.10.2023.
[14]. Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms, Available at: https://gitlab.com/libeigen/eigen, accessed 23.10.2023.
[15]. Armadillo: C++ Library for Linear Algebra & Scientific Computing, Available at: https://gitlab.com/conradsnicta/armadillo-code, accessed 23.10.2023.
[16]. Cui E., Li T., Wei Q. RISC-V Instruction Set Architecture Extensions: A Survey. IEEE Access, 2023, pp. 1-16. DOI: 10.1109/ACCESS.2023.3246491.
[17]. The European High Performance Computing Joint Undertaking (EuroHPC JU), Available at: https://eurohpc-ju.europa.eu/index_en, accessed 23.10.2023.
[18]. Occamy, Available at: https://pulp-platform.org/occamy/, accessed 23.10.2023.
[19]. The EUPILOT Project, Available at: https://eupilot.eu/, accessed 23.10.2023.
[20]. PRIMME: PReconditioned Iterative MultiMethod Eigensolver, Available at: https://github.com/primme/primme, accessed 23.10.2023.
[21]. BLAS-Tester, Available at: https://github.com/xianyi/BLAS-Tester, accessed 23.10.2023.
[22]. T-HEAD GNU Compiler Toolchain, Available at: https://github.com/T-head-Semi/xuantie-gnu-toolchain, accessed 23.10.2023.
[23]. LCOV, Available at: https://github.com/linux-test-project/lcov, accessed 23.10.2023.

Информация об авторах / Information about authors

Ксения Алексеевна Зайцева является младшим инженером-программистом Департамента разработки высокопроизводительных библиотек компании YADRO. Ее научные интересы включают низкоуровневые оптимизации, генераторы случайных чисел.

Ksenia Alexeyevna Zaytseva is a junior software engineer of the Department of High Performance Libraries Development of YADRO. Her research interests include low-level optimizations, random number generators.

Валерия Валентиновна ПУЗИКОВА – кандидат физико-математических наук, эксперт по разработке программного обеспечения Департамента разработки высокопроизводительных библиотек компании YADRO. Сфера научных интересов: решатели и предобуславливатели СЛАУ, разработка прикладных математических программ, вычислительная гидродинамика, физические движки для AR/VR, высокопроизводительные вычисления, численные методы.

Valeria Valentinovna PUZIKOVA – Cand. Sci. (Phys.-Math.), Software Development Expert in High Performance Libraries Department, YADRO. Research interests: solvers and preconditioners for SLAE, applied mathematics software development, computational hydrodynamics, physics engines for AR/VR, high performance computations, numerical methods.

Андрей Дмитриевич СОКОЛОВ является инженером-программистом Департамента разработки высокопроизводительных библиотек компании YADRO. Его научные интересы включают разработку и тестирование математических библиотек, низкоуровневые оптимизации, вычисления на GPU, сверточные нейронные сети.

Andrey Dmitrievich SOKOLOV is a software engineer of the Department of High Performance Libraries Development of YADRO. His research interests include development and testing of mathematical libraries, low-level optimizations, GPU computing, convolutional neural networks.