2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

doi 10.26089/NumMet.v26r430 УДК 04.272, 004.4'242

Применение многоуровневого параллелизма в программах, использующих параллельные библиотеки

В. А. Бахтин
Институт прикладной математики имени М. В. Келдыша (ИПМ РАН),
Москва, Российская Федерация
ORCID: 0000-0003-0343-3859, e-mail: bakhtin@keldysh.ru

Н. А. Катаев
Институт прикладной математики имени М. В. Келдыша (ИПМ РАН),
Москва, Российская Федерация
ORCID: 0000-0002-7603-4026, e-mail: kataev@keldysh.ru

А. С. Колганов
Институт прикладной математики имени М. В. Келдыша (ИПМ РАН),
Москва, Российская Федерация
ORCID: 0000-0002-1384-7484, e-mail: alexander.k.s@mail.ru

Д. А. Захаров
Институт прикладной математики имени М. В. Келдыша (ИПМ РАН),
Москва, Российская Федерация
ORCID: 0000-0002-6319-5090, e-mail: s123-93@mail.ru

А. А. Смирнов
Институт прикладной математики имени М. В. Келдыша (ИПМ РАН),
Москва, Российская Федерация
ORCID: 0000-0002-2971-4248, e-mail: smiraland@gmail.com

А. А. Малахов
Автономная некоммерческая организация высшего образования "Университет НЕЙМАРК",
Нижний Новгород, Российская Федерация
ORCID: 0009-0004-9276-2172, e-mail: A.a.m@inbox.ru

Аннотация: С ростом числа вычислительных ядер и использующих их потоков возрастают накладные расходы на их планирование, порождение и уничтожение, а объем вычислений, выполняемых каждым потоком в отдельности, сокращается. Многоуровневый параллелизм — способ обойти данную проблему. Его источником может стать распараллеливание функций, реализованных внутри библиотек, вызываемых из параллельной программы. Данный подход может потребовать дополнительной поддержки на уровне моделей параллельного программирования. В нашем исследовании мы предлагаем альтернативный способ реализации библиотеки времени выполнения OpenMP и исследуем его эффективность на примере библиотеки OpenBLAS и бенчмарка High Performance Linpack для Arm-систем.

Ключевые слова: мультипроцессор, многоуровневый параллелизм, пользовательские потоки, OpenMP, TBB, OpenBLAS, HPL.

Для цитирования: Бахтин В.А., Катаев Н.А., Колганов А.С., Захаров Д.А., Смирнов А.А., Малахов А.А. Применение многоуровневого параллелизма в программах, использующих параллельные библиотеки // Вычислительные методы и программирование. 2025. **26**, № 4. 449–464. doi 10.26089/NumMet.v26r430.

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

Applying composable parallelism in programs using parallel libraries

Vladimir A. Bakhtin
Keldysh Institute of Applied Mathematics of RAS, Moscow, Russia
ORCID: 0000-0003-0343-3859, e-mail: bakhtin@keldysh.ru

Nikita A. Kataev
Keldysh Institute of Applied Mathematics of RAS, Moscow, Russia
ORCID: 0000-0002-7603-4026, e-mail: kataev@keldysh.ru

Alexander S. Kolganov
Keldysh Institute of Applied Mathematics of RAS, Moscow, Russia
ORCID: 0000-0002-1384-7484, e-mail: alexander.k.s@mail.ru

Dmitriy A. Zakharov
Keldysh Institute of Applied Mathematics of RAS, Moscow, Russia
ORCID: 0000-0002-6319-5090, e-mail: s123-93@mail.ru

Alexander A. Smirnov
Keldysh Institute of Applied Mathematics of RAS, Moscow, Russia
ORCID: 0000-0002-2971-4248, e-mail: smiraland@gmail.com

Anton A. Malakhov
Autonomous Non-profit Organization of Higher Education "NEIMARK University", Nizhny Novgorod, Russia
ORCID: 0009-0004-9276-2172, e-mail: A.a.m@inbox.ru

Abstract: As the number of computational cores and the threads utilizing them increases, the overhead associated with thread scheduling, creation, and destruction grows, while the amount of computation performed by each individual thread decreases. Multilevel parallelism is a way to address this issue. One potential source of such parallelism is the optimization of functions within libraries called from parallel programs. However, this approach may require additional support at the level of parallel programming models. In our study, we propose an alternative implementation of the OpenMP runtime library and evaluate its efficiency using the OpenBLAS library and the High Performance Linpack (HPL) benchmark on Arm-based systems.

Keywords: multiprocessor, multilevel parallelism, user threads, OpenMP, TBB, OpenBLAS, HPL.

For citation: V. A. Bakhtin, N. A. Kataev, A. S. Kolganov, D. A. Zakharov, A. A. Smirnov, A. A. Malakhov, "Applying composable parallelism in programs using parallel libraries," Numerical Methods and Programming. **26** (4), 449–464 (2025). doi 10.26089/NumMet.v26r430.

1. Введение. За прошедшее десятилетие наблюдается значительный рост числа процессорных ядер, входящих в состав современных вычислительных систем. Если раньше широкое распространение получали системы, использующие до десяти процессорных ядер, то сейчас число ядер в системах на общей памяти может насчитывать несколько сотен. Такой стремительный рост сложности вычислительных систем является вызовом для разработчиков прикладных программ и алгоритмов с точки зрения эффективного использования всех имеющихся вычислительных ресурсов. Невозможность постоянного уменьшения порций работ, выполняемых параллельно, из-за возникающих накладных расходов на порождение, уничтожение и планирование отдельной порции ведет к необходимости поиска дополнительных источников параллелизма. Вложенный параллелизм [1–5] — один из возможных путей решения.

OpenMP — общепризнанный стандарт для разработки параллельных программ для систем с общей памятью, поддерживающий языки программирования C/C++ и Fortran, активно применяющиеся при разработке высокопроизводительных приложений. Кроме того, высокий уровень конструкций OpenMP, представляющих собой директивные расширения поддерживаемых языков, и инкрементальный способ

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

распараллеливания уже имеющихся последовательных программ значительно снижают порог вхождения в область разработки параллельных программ.

OpenMP позволяет описать параллелизм явно на основе выделения потоков, при этом стандартная реализация системы поддержки времени выполнения (GCC, LLVM) полагается на использование потоков уровня операционной системы. Таким образом, при том что OpenMP явно предусматривает возможность использования многоуровневого параллелизма, эффективность программ, использующих более одного уровня, без ручного подбора количества потоков на каждом уровне может приводить к деградации производительности из-за превышения количества создаваемых потоков уровня операционной системы над количеством физически доступных вычислительных ресурсов.

Некоторые реализации системы поддержки времени выполнения позволяют ограничить число одновременно исполняемых параллельных областей за счет реализации специального механизма блокировок [2]. Соответствующий режим исполнения OpenMP программ может быть задействован за счет использования дополнительных переменных окружения (например `KMP_COMPOSABILITY`), предоставляемых конкретной реализацией библиотеки.

Другим возможным решением в данной ситуации может быть добавление промежуточного уровня абстракции в виде потоков уровня пользователя, которые бы позволили отобразить несколько потоков на один поток уровня операционной системы и избавиться от несоответствия доступных физических ресурсов количеству запрашиваемых потоков уровня операционной системы. Одной из реализаций OpenMP, направленных на решение данной проблемы, является BOLT [6]. Данная реализация опирается на легковесную систему времени выполнения на основе пользовательских потоков Argobots [7]. Недостатком BOLT является деградация производительности при использовании подхода классического параллелизма[1] [8] по сравнению со стандартными реализациями OpenMP, что является следствием дополнительных расходов, возникающих во время планирования выполнения пользовательских потоков.

Использование большого числа даже пользовательских потоков требует реализации эффективных алгоритмов динамической балансировки нагрузки, мелкозернистых синхронизаций и планировщика, способного адаптироваться под архитектуру вычислительной системы и особенности конкретной программы. Достаточно гибкие средства планирования задач реализованы в библиотеке oneAPI Threading Building Blocks (TBB) [9] — высокоуровневой C++ библиотеке, которая предоставляет разработчику средства разработки параллельных программ, основанных на явном использовании параллелизма задач. Данная библиотека может являться одним из возможных кандидатов для реализации предложенного выше уровня абстракции между высокоуровневой моделью параллельного программирования и множеством потоков уровня операционной системы.

Исследованиям вложенного параллелизма уделяется большое внимание в последнее время. Создаются различные библиотеки легковесных потоков (lightweight threads) [7, 10, 11], позволяющие более полно использовать ресурсы центрального процессора. Предлагаются различные стратегии распределения работ по потокам и способы балансировки нагрузки [12–15]. Исследуются модели отображения потоков “многие-ко-многим” ($M : N$ потоки), где M пользовательских потоков, видимых программисту, отображаются на N потоков уровня ядра операционной системы, что позволяет выполнять переключение между потоками и планирование почти полностью в пользовательском пространстве без участия ядра операционной системы. Многие современные реализации $M : N$ потоков, такие как Argobots, Qthreads и MassiveThreads, сокращают накладные расходы на переключение контекста потоков по сравнению со стандартной реализацией, использующей Pthreads, что позволяет использовать более мелкозернистый параллелизм и разрабатывать оптимизированные для него планировщики [16].

В данной работе мы рассматриваем влияние вложенного параллелизма на эффективность программ, использующих библиотеки для ускорения отдельных этапов вычислений. Внимание уделяется совместному использованию параллелизма, реализованного в прикладной программе пользователя, и параллельной реализации используемой библиотеки. Исследование направлено на рассмотрение реализаций библиотеки базовых подпрограмм линейной алгебры (BLAS), особое внимание уделяется реализации OpenBLAS [20].

Среди основных результатов данной работы можно выделить разработку прототипа системы компиляции, эффективно реализующей подмножество конструкций OpenMP, которые отображаются в вызовы библиотеки поддержки параллельного выполнения программы. Разработанная библиотека поддержки

[1] Подход к разработке параллельных приложений с одним уровнем параллелизма, основанный на использовании потоков уровня операционной системы.

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

времени выполнения основана на применении промежуточного уровня абстракции в виде пользовательских потоков для оптимизации параллельного выполнения большого количества потоков, общее число которых может превышать физически доступные вычислительные ресурсы.

Статья организована следующим образом. В разделе 2 рассматривается библиотека BLAS и некоторые из ее существующих реализаций, обосновывается выбор в пользу реализации OpenBLAS, как предмета данного исследования. В разделе 3 рассматривается реализованный прототип системы компиляции и обсуждаются особенности реализации библиотеки времени выполнения в случае использования ее для параллельного выполнения библиотек. Внимание также уделяется хранению контекста, описывающего структуру выполняющегося пользовательского потока, в условиях оптимизаций обращений к памяти из разных потоков операционной системы, выполняемых стандартными компиляторами. В разделе 4 приводятся результаты экспериментов, предусматривающих использование библиотеки OpenBLAS в условиях вложенного параллелизма, а также исследуется потенциальный выигрыш, которого можно достичь при выполнении бенчмарка High Performance Linpack (HPL). В разделе 5 кратко сформулированы результаты проведенного исследования.

2. Операции линейной алгебры. Пакет линейной алгебры (LAPACK) — де-факто стандарт, который принят и поддержан большим сообществом пользователей, вычислительными центрами и поставщиками оборудования для высокопроизводительных вычислений. Этот пакет включает в себя самые современные численные алгоритмы для наиболее распространенных задач линейной алгебры, встречающихся в научных и инженерных приложениях: решение линейных уравнений, задач на собственные значения и др. Пакет LAPACK построен на базе BLAS.

Библиотека BLAS является одной из наиболее важных базовых программных библиотек в научных вычислениях. Эталонная реализация библиотеки реализована на Fortran [17]. В нее вошли процедуры, реализующие следующие преобразования:

- скалярное произведение двух векторов;
- копирование элементов одного вектора в другой;
- перестановка местами элементов двух векторов;
- умножение вектора на скаляр;
- векторные нормы (евклидова норма, сумма абсолютных значений элементов и максимальное абсолютное значение);
- другие процедуры типа “вектор-вектор”, “матрица-вектор”, “матрица-матрица”.

Помимо этой реализации, существуют оптимизированные варианты практически для каждой компьютерной архитектуры. Эти реализации в основном предоставляются производителями процессоров или аппаратного обеспечения для получения максимальной производительности на этих устройствах [18, 19]. В дополнение к реализациям от поставщиков, существуют реализации с открытым исходным кодом, такие как OpenBLAS [20] и Automatically Tuned Linear Algebra Software (ATLAS) [21].

Преимуществами библиотеки OpenBLAS являются:

- библиотека активно развивается (последняя версия библиотеки 0.3.29 разработана в январе 2025 г.);
- широкий спектр поддерживаемых архитектур: RISC-V, X64, SPARC, X86, MIPS, ARM, AArch64, POWER, PPC64, IBM System z;
- существует параллельная реализация библиотеки с помощью POSIX Threads и OpenMP;
- библиотека полностью реализует интерфейсы BLAS и LAPACK, а также предоставляет дополнительное множество BLAS-like функций для расширения возможностей BLAS;
- BLAS-часть библиотеки содержит в себе набор базовых операций линейной алгебры, которые чаще всего используются в прикладных программах. Все функции оптимизированы как алгоритмически, так и на низком уровне, а на их основе строятся реализации более сложных алгоритмов, входящих в LAPACK;
- поддерживается большим сообществом, в составе которого не только независимые разработчики, но и крупные компании;
- наличие в библиотеке встроенных тестов, которые позволяют сравнивать различные реализации BLAS-библиотек по их эффективности. Некоторые тесты распараллелены с использованием OpenMP;
- один из основных модулей библиотеки, который используется для распараллеливания вычислений (blas_server_omp.c), состоит всего из 458 строк, что позволяет, в случае необходимости, адаптировать его в соответствии с возможностями, реализованными в разработанной версии компилятора.

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

Как уже отмечалось ранее, в составе OpenBLAS есть встроенные тесты, позволяющие оценить эффективность реализации различных функций библиотеки. Эти тесты, в основном, построены следующим образом. В последовательном цикле перебираются необходимые входные данные, задаются размеры матриц, определяется количество итераций алгоритма и т.п. Далее замеряется время работы, а также производительность программы (в MFlops). Такой поход позволяет сравнить качество реализации функций библиотеки при решении различных задач на разных поддерживаемых библиотекой архитектурах.

Фрагмент программы, представленный в листинге 1, используется при тестировании для процедуры `SYMM`, выполняющей преобразование вида `C=alpha*A*B+beta*C` или `C=alpha*B*A+beta*C`, где `alpha` и `beta` — скалярные переменные, матрица `A` — симметричная матрица размера $m \times m$ или $n \times n$, `B` и `C` — матрицы размера $m \times n$ (symm.c).

Листинг 1. Фрагмент программы, используемой при тестировании для процедуры `SYMM`
Listing 1. A fragment of the program used in testing for the `SYMM` procedure

```
for(m = from; m <= to; m += step) {
  for(j = 0; j < m; j++) {
    for(i = 0; i < m * COMPSIZE; i++) {
      A[(long)i+(long)j*(long)m*COMPSIZE] = ((FLOAT)rand()/(FLOAT)RAND_MAX)-0.5;
      B[(long)i+(long)j*(long)m*COMPSIZE] = ((FLOAT)rand()/(FLOAT)RAND_MAX)-0.5;
      C[(long)i+(long)j*(long)m*COMPSIZE] = ((FLOAT)rand()/(FLOAT)RAND_MAX)-0.5;
    }
  }
  begin();
    SYMM (&side, &uplo, &m, &m, alpha, A, &m, B, &m, beta, C, &m );
  end();
  time1 = getsec();
  fprintf(stderr, "%10.2f␣MFlops\n",
    COMPSIZE*COMPSIZE*2.*(double)m*(double)m*(double)m/time1*1.e-6);
}
```

Все матрицы при тестировании считаются квадратными (размера $m \times m$). Размер матриц, на которых запускается процедура, перебирается в цикле. Сначала выполняется инициализация матриц `A`, `B` и `C` случайными значениями, а затем вызывается процедура `SYMM`, выполняющая необходимое преобразование. Замеряется время ее работы. Такое тестирование выполняется для матриц, использующих разные типы данных: `float`, `double`, `complex float`, `complex double`.

3. Прототип системы компиляции. Данное исследование предполагает разработку нового подхода, использующего абстракцию в виде пользовательских потоков, к реализации библиотеки времени выполнения для высокоуровневых моделей параллельного программирования. Таким образом, для достижения целей данного исследования был разработан транслятор подмножества спецификаций стандарта OpenMP, выполняющий конвертацию OpenMP-программы в программу на языке высокого уровня с вызовами библиотеки времени выполнения. Подмножество спецификаций выбиралось так, чтобы его было достаточно для описания параллелизма в исследуемых приложениях. Библиотека времени выполнения, контролирующая выполнение параллельной программы, также была разработана в рамках данного исследования.

Использование высокоуровневых спецификаций OpenMP для эффективного описания многоуровневого параллелизма вызывает отдельный интерес, так как способствует выработке рекомендаций по написанию и модификации большого числа уже написанных в модели OpenMP программ.

Библиотека времени выполнения обеспечивает промежуточный интерфейс для использования различных технологий параллельного программирования (OpenMP Task, OpenMP Taskloop, Argobots [7], Taskflow [22], TBB [9]) в зависимости от разных вариантов ее реализации [5, 8]. В данной работе основное внимание уделяется подходу, основанному на использовании возможностей TBB в сравнении со стандартными возможностями, предоставляемыми OpenMP для описания параллелизма уровня циклов.

Было реализовано два типа пользовательских потоков, различающихся в зависимости от выбора конструкций OpenMP их порождающих. Обработка директивы `omp parallel for` отличается от обработки директив `parallel` и `for`, указываемых по отдельности, так как при совместном их указании компи-

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

лятор и библиотека времени выполнения обладают большей информацией о параллельно выполняемых вычислениях и могут определять количество создаваемых легких потоков в зависимости от количества итераций в параллельном цикле. Если же директивы указаны по отдельности, то на момент порождения потоков, а именно обработки директивы `parallel`, невозможно определить оптимальное число потоков для выполнения параллельной области и приходится использовать заранее заданные настройки по умолчанию.

Для разработки транслятора были использованы существующие средства, предоставляемые инфраструктурой компиляторов LLVM [23] и имеющимися в ней API с языками программирования высокого уровня.

Для упрощения компиляции и запуска программ данные этапы объединены под управлением отдельного инструмента — драйвера. Настройки компиляции и запуска задаются как через опции драйвера, так и через переменные окружения, которые позволяют управлять числом потоков уровня операционной системы, ограничениями на максимальное число порождаемых пользовательских потоков, используемыми алгоритмами динамического планирования и балансировки нагрузки, а также способами распределения потоков по NUMA-узлам вычислительной системы.

3.1. Особенности конвертации внутри библиотек. Для приложений, использующих библиотеку OpenBLAS, специфично обращение к некоторым функциям OpenMP до начала выполнения функции `main()` в момент загрузки динамической библиотеки, реализующей функции OpenBLAS. Для успешного выполнения этих функций необходима предварительная инициализация библиотеки времени выполнения OpenMP. Для реализованных версий библиотеки такая инициализация выполняется специальной функцией инициализации, вызов которой вставляется в начало функции `main`.

Чтобы выполнить инициализацию до начала исполнения `main`, был использован механизм атрибутов функций для задания функций, вызываемых в момент загрузки динамической библиотеки: `__attribute__ ((constructor(101)))`. Кроме того, для данных функций были явно заданы приоритеты исполнения, чтобы обеспечить их вызов до начала исполнения функций OpenBLAS.

Аналогичные изменения были внесены для высвобождения ресурсов внутри соответствующих функций, помеченных атрибутом `__attribute__((destructor(101)))`.

3.2. Особенности хранения контекста пользовательских потоков. Для того чтобы иметь возможность в любой точке программы идентифицировать пользовательский поток, который ее обрабатывает, необходимо сохранить информацию о потоке в глобальной области видимости. Так как одновременно могут выполняться несколько пользовательских потоков, каждый из которых выполняется своим потоком операционной системы, то для хранения контекста выполнения потока был использован подход с применением thread-local-storage (TLS) (листинг 2).

Таким образом, перед началом выполнения пользовательского потока текущее значение контекста из TLS потока уровня операционной системы запоминается в локальной переменной, а после завершения потока в TLS восстанавливается информация о родительском потоке.

Листинг 2. Сохранение контекста выполнения пользовательского потока в TLS потоке уровня операционной системы

Listing 2. Saving the execution context of a user thread to the TLS of an operating system-level thread

```
1  static thread_local context_t *this_context = nullptr;
2
3  context_t * get_context() {
4    return this_context;
5  }
6
7  void set_context(context_t *context) {
8    this_context = context;
9  }
```

Стоит отметить, что при определенных обстоятельствах, которые могут проявляться из-за недетерминированности выполнения пользовательских потоков на потоках операционной системы, поток операционной системы, который выполнял родительский пользовательский поток до начала выполнения дочер-

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

него пользовательского потока, может смениться после завершения дочернего пользовательского потока. Например, в зависимости от особенностей выполнения потоков в листинге 3 поток операционной системы до начала выполнения внутреннего параллельного цикла мог иметь идентификатор `TID_1`, а после его завершения `TID_2`. Таким образом, TLS до и после выполнения внутреннего цикла — различны.

Особенность состоит в том, что если объявление TLS переменной видимо для стандартного компилятора в момент оптимизации кода для данного фрагмента, то компилятор может предположить, что данное значение осталось неизменным, так как в коде программы отсутствуют явные указания на возможность его изменения (стандартный компилятор не знает о пользовательских потоках и о том, что они могут менять принадлежность к потоку операционной системы, который их исполняет). Таким образом, компилятор может заменить в коде программы все использования `TID_2` на использования `TID_1` в целях оптимизации, что приведет к некорректной программе.

Чтобы обойти данную проблему, обращение к TLS переменным было вынесено в специальные функции, приведенные в листинге 2, а сами функции были вынесены в отдельную единицу компиляции. При этом важно, чтобы в момент сборки программы была отключена межмодульная оптимизация и тела данных функций не были подставлены в точки вызова до и после момента исполнения внутреннего цикла в листинге 3.

Листинг 3. Потенциальная возможность некорректной оптимизации кода стандартными компиляторами в случае использования TLS

Listing 3. Potential for incorrect code optimization by standard compilers when using TLS

```
#pragma omp parallel for
for (int i = 0; i < N; ++i) {
  // Some work in outer user-level thread.
  // Let us assume OS-level thread at this point is TID_1.
  #pragma omp parallel for
  for (int j = 0; j < N; ++j) {
    // Some work in inner user-level thread.
  }
  // Let us assume OS-level thread at this point is TID_2.
  // At this point TID_1 may differ from TID_2.
}
```

4. Эксперименты. В рамках данного исследования эксперименты проводились на следующих вычислительных системах:

- сервер с процессорами Kunpeng 2.6 ГГц (архитектура aarch64), включающий 192 ядра (8 NUMA узлов — 24 ядра на узел), 1 поток на ядро, 503 ГБ, Ubuntu 18.04.6 LTS (далее — сервер Atlas),
- сервер с процессорами Kunpeng 920 2.6 ГГц (архитектура aarch64), включающий 48 ядер (2 NUMA узла — 24 ядра на узел), 1 поток на ядро, 512 ГБ, CentOS Linux 8 (далее — сервер Kunpeng).

В исследовании использовался вычислительный кластер на основе Huawei TaiShan 200 ЦКП ССКЦ ИВМиМГ СО РАН.

Чтобы обеспечить одинаковое программное окружение для запуска приложений, был использован дистрибутив Miniconda [24], включающий в себя систему управления пакетами и средой выполнения приложений. Для компиляции приложений был использован компилятор GCC 13.2.0 и TBB 2021.11, при компиляции указывалась опция оптимизации `-O3`.

Сборка библиотеки OpenBLAS выполнялась с настройками по умолчанию, заданными в конфигурационных файлах библиотеки в зависимости от целевой платформы. Основное внимание в экспериментах уделялось влиянию вложенного параллелизма на производительность программ, поэтому дополнительная оптимизация вычислительных ядер не проводилась, также предполагалось, что за векторизацию кода для конкретной целевой платформы отвечает OpenBLAS.

4.1. Библиотека OpenBLAS. Для исследования влияния многоуровневого параллелизма на эффективность параллельного выполнения программ, использующих библиотеку OpenBLAS, было разработано несколько тестовых приложений, реализующих умножение различных матриц, решающих различные матричные уравнения. Разработанный пакет тестов позволяет задействовать и исследовать два уровня па-

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

Листинг 4. Фрагмент параллельной программы, используемой при тестировании процедуры SYMM и задействующей два уровня параллелизма

Listing 4. A fragment of a parallel program used to test the SYMM procedure and employing two levels of parallelism

```
omp_set_num_threads(num_threads_in);
#pragma omp parallel for private(A,B,C,aa,bb,cc,i,j) \
                         num_threads(num_threads_out) schedule(runtime)
for(m = from; m <= to; m += step) {
  A = a1[m - from];
  B = b1[m - from];
  C = c1[m - from];
  aa = a0[m - from];
  bb = b0[m - from];
  cc = c0[m - from];
  for(j = 0; j < m; j++) {
    for(i = 0; i < m * COMPSIZE; i++) {
      A[(long)i+(long)j*(long)m*COMPSIZE] = aa[(long)i+(long)j*(long)m*COMPSIZE];
      B[(long)i+(long)j*(long)m*COMPSIZE] = bb[(long)i+(long)j*(long)m*COMPSIZE];
      C[(long)i+(long)j*(long)m*COMPSIZE] = cc[(long)i+(long)j*(long)m*COMPSIZE];
    }
  }
}
time1 = omp_get_wtime();
#pragma omp parallel for private(A,B,C) \
                         num_threads(num_threads_out) schedule(runtime)
for(m = from; m <= to; m += step) {
  A = a1[m - from];
  B = b1[m - from];
  C = c1[m - from];
  SYMM (&side, &uplo, &m, &m, alpha, A, &m, B, &m, beta, C, &m );
}
time1 = omp_get_wtime() - time1;
fprintf(stderr, "|␣␣␣␣␣␣␣␣␣␣␣␣%3d|␣␣␣␣␣␣␣␣␣␣␣␣%3d|%11.6f|\n",
        num_threads_out, num_threads_in, time1);
```

раллелизма — в основной программе и внутри библиотеки OpenBLAS. Указанный в листинге 1 фрагмент программы был переписан так, как показано в листинге 4.

При помощи процедуры `omp_set_num_threads(num_threads_in)` задается количество потоков, которые будут использоваться библиотекой OpenBLAS для выполнения умножения матриц. Параллельный цикл `for(m = from; m <= to; m += step)` запускает на выполнение множество задач, каждая из которых выполняет требуемое `SYMM`-преобразование для матриц соответствующего размера $m \times m$. Количество и сложность решаемых задач задается при помощи параметров цикла `from`, `to` и `step`. Необходимые для работы программы данные инициализируются один раз, а затем копируются из соответствующих массивов перед выполнением преобразования каждый раз (вместо вызовов генератора случайных чисел) для исключения влияния используемых в процессе счета данных(чисел) на время выполнения программы.

Такой подход позволяет многократно запускать различные преобразования, задавая различное число потоков на внешнем (`num_threads_out`) и внутреннем (`num_threads_in`) уровнях, меняя стратегии распределения работ по потокам (спецификация `schedule`), варьируя количество выполняемых задач и изменяя их сложность.

В табл. 1 представлены результаты такого исследования теста `SYMM` на сервере Atlas при использовании до 192 OpenMP-потоков (используется реализация OpenMP, предоставляемая компилятором GCC). В таблице для разных типов данных отдельно выделено наименьшее время и наибольшее ускорение относительно выполнения программы на одном потоке.

В табл. 2 представлены результаты для теста `HEMM`, решающего аналогичную задачу, в которой `A` — эрмитова матрица. Элементы этой матрицы являются комплексными числами, и, будучи транспонирована, она равна комплексно сопряженной.

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

Таблица 1. Время и ускорение, полученные при выполнении теста `SYMM` на разном числе потоков для конфигурации: `from=1400, to=1800, step=1, schedule=dynamic`

Table 1. `SYMM` test execution times and speedups on different numbers of threads for the configuration: `from=1400, to=1800, step=1, schedule=dynamic`

Конфигурация потоков Threads configuration	Тип данных Data type							
	`float`		`double`		`complex float`		`complex double`	
	Время, с Time, s	Ускорение Speedup	Время, с Time, s	Ускорение Speedup	Время, с Time, s	Ускорение Speedup	Время, с Time, s	Ускорение Speedup
1×1	124.8	1.00	339.1835	1.00	517.19	1.00	1918.4	1.00
1×192	4.74	26.34	8.1486	41.62	16.83	30.73	33.61	57.08
6×32	2.20	56.69	4.5106	75.20	4.22	122.54	11.52	166.49
12×16	**1.44**	**86.79**	3.49	97.58	**3.78**	**136.87**	**11.23**	**170.85**
24×8	1.54	81.27	3.91	86.76	4.08	126.74	12.33	155.58
48×4	2.20	56.62	**3.47**	**97.69**	4.42	116.96	12.49	153.56
96×2	2.23	55.89	3.62	93.66	4.35	118.90	12.82	149.66
192×1	2.78	44.74	4.49	75.62	5.02	103.01	15.34	125.09

Таблица 2. Время и ускорение, полученные при выполнении теста `HEMM` на разном числе потоков для конфигурации: `from=1400, to=1800, step=1, schedule=dynamic`

Table 2. `HEMM` test execution times and speedups on different numbers of threads for the configuration: `from=1400, to=1800, step=1, schedule=dynamic`

Конфигурация потоков Threads configuration	Тип данных Data type			
	`complex float`		`complex double`	
	Время, с Time, s	Ускорение Speedup	Время, с Time, s	Ускорение Speedup
1×1	516.26	1.00	1914.29	1.00
1×192	16.71	30.89	26.63	71.87
6×32	4.77	108.21	11.69	163.73
12×16	**4.01**	**128.74**	**11.4279**	**167.51**
24×8	4.13	125.05	11.91	160.70
48×4	4.27	121.02	12.0356	159.05
96×2	4.36	118.53	12.4746	153.46
192×1	4.88	105.83	15.0869	126.88

В табл. 3 представлены результаты для теста `TRSM`, который предназначен для решения левостороннего `A*X=alpha*Y` или правостороннего `X*A=alpha*Y` матричного уравнения для всевозможных вариантов представления треугольной матрицы `A`. Установкой параметров можно задать не только, с какой стороны в этом уравнении участвует матрица `A`, но и уточнить, является ли она нижней или верхней треугольной, транспонированная ли она или сопряженная (для комплексных), единичная ли у нее диагональ или нет.

В табл. 4 приведены значения наилучшего времени выполнения различных тестов при использовании 192 потоков на сервере Atlas, когда распараллелен только цикл, осуществляющий запуск множества задач, а параллелизм внутри OpenBLAS не используется (столбец “Внешний”), когда цикл по задачам выполняется последовательно, но используется параллелизм внутри OpenBLAS (столбец “Внутренний”), а также наилучший запуск, который использует два уровня параллелизма (столбец “Вложенный”). В таблице также представлен прирост производительности, получаемый за счет использования вложенного параллелизма по отношению к одному уровню параллелизма (всегда выбирался внешний параллелизм, так как в проведенных экспериментах он показывает лучшее время по сравнению с внутренним параллелизмом).

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

Таблица 3. Время и ускорение, полученные при выполнении теста `TRSM` на разном числе потоков для конфигурации: `from=1000, to=1200, step=1, schedule=dynamic, side=L, uplo=U, trans=N, diag=U, loops=3`

Table 3. `TRSM` test execution times and speedups on different numbers of threads for the configuration: `from=1000, to=1200, step=1, schedule=dynamic, side=L, uplo=U, trans=N, diag=U, loops=3`

Конфигурация потоков Threads configuration	Тип данных Data type							
	`float`		`double`		`complex float`		`complex double`	
	Время, с Time, s	Ускорение Speedup	Время, с Time, s	Ускорение Speedup	Время, с Time, s	Ускорение Speedup	Время, с Time, s	Ускорение Speedup
1×1	40.23	1.00	89.27	1.00	140.60	1.00	468.58	1.00
1×192	11.43	3.52	18.56	4.81	19.13	7.35	22.76	20.58
6×32	4.64	8.66	2.71	32.96	2.89	48.66	4.94	94.88
12×16	1.83	21.93	1.59	56.10	1.75	80.13	4.29	109.31
24×8	1.0968	36.69	1.44	62.01	1.8	78.26	4.65	100.76
48×4	**0.78**	**51.26**	1.39	64.23	1.77	79.55	**4.12**	**113.86**
96×2	0.99	40.56	**1.28**	**69.77**	**1.74**	**80.70**	4.23	110.79
192×1	0.84	47.71	1.31	68.24	1.95	72.27	5.44	86.16

Таблица 4. Значения наилучшего времени выполнения на сервере Atlas тестов `SYMM`, `HEMM`, `TRSM` при 192 потоках с использованием разных уровней параллелизма для конфигураций тестов из табл. 1–3 соответственно и типах данных `float`, `double`, `complex float`, `complex double`. Ускорение рассчитано как отношение времени выполнения реализации с использованием вложенного параллелизма к лучшему времени выполнения реализации, использующей один уровень параллелизма

Table 4. Best execution times on the Atlas server for `SYMM`, `HEMM`, `TRSM` tests on 192 threads using different parallelism levels for test configurations from tables 1–3 respectively and data types `float`, `double`, `complex float`, `complex double`. The column "Speedup" shows the speedup obtained from using nested parallelism, relative to the best option using one parallelism level

Тип данных Data type	Внешний, с External, s	Внутренний, с Internal, s	Вложенный, с Nested, s	Ускорение Speedup
`SYMM`				
`float`	2.79	4.74	**1.44**	**51%**
`double`	4.49	8.15	**3.47**	**23%**
`complex float`	5.02	16.83	**3.78**	**25%**
`complex double`	15.34	33.61	**11.23**	**27%**
`HEMM`				
`float`	—	—	—	—
`double`	—	—	—	—
`complex float`	4.88	16.71	**4.01**	**18%**
`complex double`	15.07	26.63	**11.43**	**24%**
`TRSM`				
`float`	0.84	11.43	**0.78**	**7%**
`double`	1.31	18.56	**1.28**	**2%**
`complex float`	1.95	19.13	**1.74**	**11%**
`complex double`	5.44	22.76	**4.12**	**24%**

По полученным в результате тестирования значениям времени, представленным в предыдущих таблицах, можно утверждать, что использование многоуровневого распараллеливания позволяет существенно ускорить выполнение тестов `SYMM`, `HEMM`, `TRSM` по сравнению с использованием одного уровня паралле-

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

лизма. Разработанные тесты позволяют управлять количеством задач, которые выполняются параллельно. Каждая задача выполняется для матриц различного размера и занимает разное время. Несмотря на использование в программах спецификации `schedule(dynamic)`, которая позволяет распределять задачи по потокам в процессе выполнения цикла (поток, раньше остальных завершивший выполнение своей задачи, получает следующую задачу на выполнение), для данных тестов не удается сбалансировать вычислительную нагрузку по потокам. Например, если выполняемых задач всего 200, а потоков — 192, то некоторым потокам приходится выполнять несколько задач, что приводит к итоговому замедлению выполнения программы. За счет включения внутреннего уровня параллелизма удается ускорить выполнение задач, что позволяет улучшить балансировку нагрузки для каждого из представленных тестов.

4.2. High Performance Linpack. Важным тестом, использующим библиотеку BLAS, является High Performance Linpack (HPL). Данный тест используется для составления рейтинга самых высокопроизводительных систем мира — TOP500. Тест HPL устроен следующим образом. Генерируется система линейных уравнений порядка n, которая решается с помощью LU-разложения с частичным поворотом строк. Для работы программы требуются библиотеки MPI и BLAS. При помощи MPI осуществляется циклическое распределение данных (2D блоки) и вычислений по процессам. Второй уровень распараллеливания возможен за счет подключения потоков при выполнении BLAS-функций.

На сервере Kunpeng было проведено сравнение различных версий теста HPL при использовании различного числа MPI-процессов и потоков, порождаемых внутри этих процессов. При выполнении данного исследования использовалась библиотека OpenMPI и различные версии библиотеки BLAS (техническая возможность установки различных версий библиотеки на данную вычислительную систему определила выбор в пользу сервера Kunpeng в дальнейших экспериментах):

- Исходная версия библиотеки OpenBLAS, скомпилированная в режиме OpenMP (далее OpenMP-версия).
- Версия библиотеки OpenBLAS, полученная при помощи компилятора, разработанного в рамках данного исследования, и использующая реализацию библиотеки времени выполнения OpenMP на основе TBB (далее TBB-версия).
- Kunpeng Math Library, входящая в пакет Kunpeng BoostKit (далее KML-версия).

Исследования проводились для разных размеров входных данных (5000×5000, 10000×10000, 20000×20000). В табл. 5–7 представлено временя выполнения различных версий теста HPL на разном числе процессов и потоков для систем линейных уравнений порядка 20000×20000. Стоит отметить, что запуски на других размерах показывают аналогичный результат, а оптимизации с точки зрения подбора размеров решаемой задачи не проводилось, так как HPL рассматривался в большей степени как одно из возможных приложений, использующих OpenBLAS. Использовались до 48 ядер сервера Kunpeng. Для OpenMPI задавались различные варианты привязки процессов (а следовательно, и потоков, порождаемых MPI-процессом) при помощи опций запуска: `-bind-to-none`, `-bind-to-numa`. В табл. 5–7 приведены значения наилучшего времени выполнения теста HPL.

Таблица 5. Время выполнения OpenMP-версии теста HPL на разном числе процессов и потоков для системы 20000×20000

Table 5. Times of execution of the OpenMP version of the HPL test on different numbers of processes and threads for the 20000×20000 system

Число MPI-процессов Number of MPIs	Количество потоков уровня операционной системы Number of OS-level threads							
	1	2	4	8	16	24	32	48
1	536.33	269.96	137.20	70.42	37.21	26.14	22.85	19.02
2	276.78	139.32	70.72	36.76	19.84	14.68		
4	141.54	72.71	37.45	20.09				
8	75.89	38.63	20.10					
16	40.06	20.69						
24	27.54	**14.44**						
32	21.36							
48	15.20							

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

Таблица 6. Время выполнения TBB-версии теста HPL на разном числе процессов и потоков для системы 20000 × 20000

Table 6. Times of execution of the TBB version of the HPL test on different numbers of processes and threads for a 20000 × 20000 system

Число MPI-процессов Number of MPIs	Количество потоков уровня операционной системы Number of OS-level threads							
	1	2	4	8	16	24	32	48
1	536.67	270.22	137.05	70.48	37.35	26.48	23.42	18.67
2	276.73	139.28	70.47	36.87	19.84	14.48		
4	141.54	72.60	37.41	19.71				
8	75.72	38.60	20.04					
16	40.05	20.61						
24	27.60	**14.58**						
32	21.16							
48	15.09							

Таблица 7. Время выполнения KML-версии теста HPL на разном числе процессов и потоков для системы 20000 × 20000

Table 7. Times of executing the KML version of the HPL test on different numbers of processes and threads for the 20000 × 20000 system

Число MPI-процессов Number of MPIs	Количество потоков уровня операционной системы Number of OS-level threads							
	1	2	4	8	16	24	32	48
1	525.62	265.76	134.70	69.23	36.93	26.37	26.50	25.19
2	271.20	136.64	69.39	36.51	20.25	15.02		
4	139.51	71.20	36.97	19.86				
8	74.43	38.00	19.82					
16	39.45	20.40						
24	27.03	**14.22**						
32	21.04							
48	14.86							

Полученные результаты показывают:

1. Использование многоуровневого параллелизма дает небольшой эффект для теста HPL. Наилучшее время выполнения теста достигается при использовании 24 MPI-процессов, в каждом из которых 2 потока. Такое поведение наблюдается практически на всех конфигурациях запуска независимо от размера решаемой задачи и используемой реализации BLAS.
2. Наилучшая конфигурация — 24 процесса по 2 потока — не позволяет в полной мере получить выигрыш от использования TBB-версии OpenBLAS по сравнению с OpenMP-версией. Одним из возможных объяснений плохого ускорения теста HPL при использовании потоков является то, что не все вычисления, выполняемые процессом, идут через OpenBLAS, часть работы выполняется MPI-процессом последовательно. Стоит также отметить, что TBB изначально написана и оптимизирована под архитектуру x86, однако показывает хорошие результаты и на архитектуре aarh64.
3. По эффективности выполнения TBB-версия теста HPL, полученная с использованием разработанного в рамках данного исследования компилятора, практически не уступает KML-версии.

5. Заключение. Проведенные исследования подтверждают, что многоуровневый параллелизм может служить источником дополнительного ускорения программ в среднем на 20–25% по сравнению с временем, затрачиваемым на выполнение программ, задействующих только один уровень параллелизма. Однако стандартные реализации общепризнанных высокоуровневых моделей параллельного программирования (таких как OpenMP) требуют кропотливой ручной настройки используемого числа потоков на

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

каждом уровне параллелизма. Отсутствие данной настройки может негативно сказаться на эффективности выполнения программы, особенно в случае превышения запрашиваемого числа потоков операционной системы над числом доступных вычислительных ядер.

С другой стороны, наши исследования показывают возможность использования единого подхода к разработке параллельных программ с разной степенью параллелизма (классический и многоуровневый параллелизм) и реализующего потенциал многоядерных процессоров для повышения производительности вычислений. Данный подход предполагает введение дополнительной абстракции для описания параллелизма в виде пользовательских потоков, используемых для отображения параллельных фрагментов программы, число которых может превышать физически доступные ресурсы, на доступные потоки операционной системы, которые затем эффективно отображаются на ядра вычислительного процессора. Более того, применение механизма пользовательских потоков может быть скрыто за высоким уровнем используемой модели параллельного программирования.

Полученные результаты открывают путь к дальнейшим исследованиям, связанным с оптимизацией планирования пользовательских потоков и определением их числа, распределением их по NUMA-узлам вычислительной системы и реализацией примитивов синхронизации. Исследование может быть полезно как для разработки стандартных реализаций моделей программирования, таких как OpenMP, оптимизируемых под различные виды целевых платформ, так и для проектирования новых дистрибутивов операционных систем.

Список литературы

1. *Fiksman E., Malakhov A.* Chapter 18 — Efficient Nested Parallelism On Large-Scale Systems // High Performance Parallelism Pearls. Eds. Reinders J., Jeffers J. Boston: Morgan Kaufmann Pub., 2015. doi 10.1016/B978-0-12-802118-7.00018-2.
2. *Malakhov A., Liu D., Gorshkov A., Wilmarth T.* Composable Multi-Threading and Multi-Processing for Numeric Libraries // Proc. 17th Python in Science Conference (SciPy 2018). 2018. 18–24. doi 10.25080/Majora-4af1f417-003.
3. *Malakhov A.* Composable Multi-Threading for Python Libraries // Proc. 15th Python in Science Conference (SciPy 2016). 2016. 15–19. doi 10.25080/Majora-629e541a-002.
4. PowerPoint Presentation - Hydra22 - Fusing - Anton Malakhov (1).pdf. https://squidex.jugru.team/api/assets/srm/2502f3b5-36bf-4e6d-88f9-8a6170b688e2/hydra22-fusing-anton-malakhov-1-.pdf. (Дата обращения: 21 октября 2025).
5. *Bakhtin V., Kataev N., Kolganov A., Zakharov D., Smirnov A., Kocharmin M.* A study of a composable approach to parallel programming for many-core multiprocessors // Proc. *Supercomputing. RuSCDays 2024. Lecture Notes in Computer Science* (Springer, Cham, 2025). Vol. 15406, 285–299. doi 10.1007/978-3-031-78459-0_21.
6. *Iwasaki S., Amer A., Taura K., et al.* BOLT: Optimizing OpenMP Parallel Regions with User-Level Threads // Proc. 28th International Conference on Parallel Architectures and Compilation Techniques (PACT '19), Sept. 2019. 29–42. doi 10.1109/PACT.2019.00011.
7. *Seo S., Amer A., Balaji P., et al.* Argobots: A Lightweight Low-Level Threading and Tasking Framework // IEEE Transactions on Parallel and Distributed Systems. 2018. **29** (3), 512–526. doi 10.1109/TPDS.2017.2766062. https://pavanbalaji.github.io/pubs/2018/tpds/tpds18.argobots.pdf. (Дата обращения: 21 октября 2025).
8. *Bakhtin V., Kataev N., Kolganov A., et al.* Exploring Composable Parallelism in Computational Modelling // Math. Models Comput. Simul. (2024). **16** (2), S216–S224. doi 10.1134/S2070048224700923.
9. GitHub - uxlfoundation/oneTBB: oneAPI Threading Building Blocks (oneTBB). https://github.com/oneapi-src/oneTBB. (Дата обращения: 21 октября 2025).
10. *Wheeler K.B., Murphy R.C., Thain D.* Qthreads: An API for programming with millions of lightweight threads // in Proc. 2008 IEEE International Symposium Parallel and Distributed Processing, Miami, FL, 2008. pp. 1–8. doi 10.1109/IPDPS.2008.4536359.
11. *Nakashima J., Taura K.* MassiveThreads: A Thread Library for High Productivity Languages // Concurrent Objects and Beyond. Lecture Notes in Computer Science. Vol. 8665. Eds. Agha G., Igarashi A., Kobayashi N., et al. Springer, Berlin, Heidelberg, 2014. doi 10.1007/978-3-662-44471-9_10.
12. *Korndörfer J.H.M., Eleliemy A., Mohammed A., Ciorba F.M.* LB4OMP: A Dynamic Load Balancing Library for Multithreaded Applications // IEEE Transactions on Parallel and Distributed Systems. 2022. **33** (4), 830–841. doi 10.1109/TPDS.2021.3107775.
13. *Ciorba F.M., Iwainsky C., Buder P.* OpenMP loop scheduling revisited: Making a case for more schedules // arXiv:1809.03188 [cs.DC]. doi 10.48550/arXiv.1809.03188.

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

14. *Penna P.H., Gomes A.T.A., Castro M., Plentz P.D.M., et al.* A comprehensive performance evaluation of the binLPT workload-aware loop scheduler // Concurrency Computation: Practice and Experience. 2019. **31** (18), e5170. doi 10.1002/cpe.5170.
15. *Kasielke F., Tschüter R., Iwainsky C., et al.* Exploring Loop Scheduling Enhancements in OpenMP: An LLVM Case Study // Proc. International Symposium on Parallel and Distributed Computing (ISPDC), Amsterdam, Netherlands. 2019. 131–138. doi 10.1109/ISPDC.2019.00026.
16. *Shiina S., Iwasaki S., Taura K., Balaji P.* Lightweight preemptive user-level threads // Proc. 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York, NY, USA. 2021. 374–388. doi 10.1145/3437801.3441610.
17. *Dongarra J.J., Du Croz J., Hammarling S., Hanson R.J.* An extended set of FORTRAN Basic Linear Algebra Subprograms // ACM Trans. Math. Softw. 1988. **14**, 1–17. doi 10.1145/42288.42291.
18. Intel® Math Kernel Library Documentation Library. https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-documentation.html. (Дата обращения: 21 октября 2025).
19. KML_BLAS Library Functions - Kunpeng BoostKit 22.0.0 Kunpeng Math Library Developer Guide 01 - Huawei. https://support.huawei.com/enterprise/en/doc/EDOC1100283141/88ccc310/kml_blas-library-functions. (Дата обращения: 21 октября 2025).
20. GitHub - OpenMathLib/OpenBLAS: OpenBLAS is an optimized BLAS library based on GotoBLAS2 1.13 BSD version. https://github.com/xianyi/OpenBLAS. (Дата обращения: 21 октября 2025).
21. *Clint Whaley R., Petitet A., Dongarra J.J.* Automated empirical optimization of software and the ATLAS project // Parallel Computing. 2001, **27** No. 1, 3–35. doi 10.1016/S0167-8191(00)00087-9.
22. Taskflow. A General-purpose Task-parallel Programming System. https://taskflow.github.io/. (Дата обращения: 21 октября 2025).
23. The LLVM Compiler Infrastructure Project. https://llvm.org/ (Дата обращения: 21 октября 2025).
24. Miniconda - Anaconda. https://www.anaconda.com/docs/getting-started/miniconda/main. (Дата обращения: 21 октября 2025).

Получена 20 августа 2025 г. | *Принята* 3 октября 2025 г. | *Опубликована* 30 октября 2025 г.

Информация об авторах

Владимир Александрович Бахтин — к.ф.-м.н., вед. науч. сотр.; Институт прикладной математики имени М. В. Келдыша (ИПМ РАН), Миусская пл., 4, 125047, Москва, Российская Федерация.

Никита Андреевич Катаев — науч. сотр.; Институт прикладной математики имени М. В. Келдыша (ИПМ РАН), Миусская пл., 4, 125047, Москва, Российская Федерация.

Александр Сергеевич Колганов — к.ф.-м.н., науч. сотр.; Институт прикладной математики имени М. В. Келдыша (ИПМ РАН), Миусская пл., 4, 125047, Москва, Российская Федерация.

Дмитрий Александрович Захаров — мл. науч. сотр.; Институт прикладной математики имени М. В. Келдыша (ИПМ РАН), Миусская пл., 4, 125047, Москва, Российская Федерация.

Александр Андреевич Смирнов — науч. сотр.; Институт прикладной математики имени М. В. Келдыша (ИПМ РАН), Миусская пл., 4, 125047, Москва, Российская Федерация.

Антон Александрович Малахов — науч. сотр.; Автономная некоммерческая организация высшего образования “Университет НЕЙМАРК”, ул. Нартова, 6, 603104, Нижний Новгород, Российская Федерация.

References

1. E. Fiksman, A. Malakhov, “Chapter 18 — Efficient Nested Parallelism On Large-Scale Systems,” in *Reinders J., Jeffers J. (eds). High Performance Parallelism Pearls.* (Morgan Kaufmann Pub., Boston, 2015). doi 10.1016/B978-0-12-802118-7.00018-2.

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

2. A. Malakhov, D. Liu, A. Gorshkov and T. Wilmarth, "Composable Multi-Threading and Multi-Processing for Numeric Libraries," in Proc. of the *17th Python in Science Conf. (SciPy 2018), Austin, Texas, USA, July 9–15, 2018,* pp. 18–24. doi 10.25080/Majora-4af1f417-003.
3. A. Malakhov, "Composable Multi-Threading for Python Libraries," in Proc. of the *15th Python in Science Conf. (SciPy 2016), Austin, Texas, USA, July 11–17, 2016,* pp. 15–19. doi 10.25080/Majora-629e541a-002.
4. PowerPoint Presentation - Hydra22 - Fusing - Anton Malakhov (1).pdf. https://squidex.jugru.team/api/assets/srm/2502f3b5-36bf-4e6d-88f9-8a6170b688e2/hydra22-fusing-anton-malakhov-1-.pdf. Cited October 21, 2025.
5. V. Bakhtin, N. Kataev, A. Kolganov, D. Zakharov, A. Smirnov, M. Kocharmin, "A Study of a Composable Approach to Parallel Programming for Many-Core Multiprocessors," in *V. Voevodin, A. Antonov, D. Nikitenko (eds) Supercomputing. RuSCDays 2024. Lecture Notes in Computer Science, Moscow, Russia, September 23–24, 2024.* (Springer, Cham, 2025). doi 10.1007/978-3-031-78459-0_21.
6. S. Iwasaki, A. Amer, K. Taura, et al., "BOLT: Optimizing OpenMP Parallel Regions with User-Level Threads," in Proc. *The 28th International Conference on Parallel Architectures and Compilation Techniques (PACT '19), Sept., 2019,* pp. 29–42. doi 10.1109/PACT.2019.00011.
7. S. Seo, A. Amer, P. Balaji, et al., "Argobots: A Lightweight Low-Level Threading and Tasking Framework," IEEE Transactions on Parallel and Distributed Systems. **29** (3), 512–526 (2018). doi 10.1109/TPDS.2017.2766062.
8. V. Bakhtin, N. Kataev, A. Kolganov, et al., "Exploring Composable Parallelism in Computational Modelling," Mathematical Models and Computer Simulations. **16** (2), S216–S224 (2024). doi 10.1134/S2070048224700923.
9. GitHub - uxlfoundation/oneTBB: oneAPI Threading Building Blocks (oneTBB). https://github.com/oneapi-src/oneTBB. Cited October 21, 2025.
10. K. B. Wheeler, R. C. Murphy, and D. Thain, "Qthreads: An API for programming with millions of lightweight threads," in Proc. *2008 IEEE International Symposium on Parallel and Distributed Processing, Miami, FL, USA, April 14–18, 2008,* pp. 1–8. doi 10.1109/IPDPS.2008.4536359.
11. J. Nakashima, K. Taura, "MassiveThreads: A Thread Library for High Productivity Languages," *In: Agha G., Igarashi A., Kobayashi N., et al. (eds). Concurrent Objects and Beyond: Lecture Notes in Computer Science. Vol. 8665.* (Springer, Berlin, Heidelberg, 2014). doi 10.1007/978-3-662-44471-9_10.
12. J. H. M. Korndörfer, A. Eleliemy, A. Mohammed, and F. M. Ciorba, "LB4OMP: A Dynamic Load Balancing Library for Multithreaded Applications," IEEE Transactions on Parallel and Distributed Systems. **33** (4), 830–841 (2022). doi 10.1109/TPDS.2021.3107775.
13. F. M. Ciorba, C. Iwainsky, and P. Buder, "OpenMP loop scheduling revisited: Making a case for more schedules," arXiv:1809.03188 [cs.DC]. doi 10.48550/arXiv.1809.03188.
14. P. H. Penna, A. T. A. Gomes, M. Castro, P. D. M. Plentz, et al., "A comprehensive performance evaluation of the binLPT workload-aware loop scheduler," Concurrency Computation: Practice and Experience, **31** (18), e5170 (2019). doi 10.1002/cpe.5170.
15. F. Kasielke, R. Tschüter, C. Iwainsky, et al., "Exploring Loop Scheduling Enhancements in OpenMP: An LLVM Case Study," in Proc. of the *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC), Amsterdam, Netherlands, 2019,* pp. 131–138, doi 10.1109/ISPDC.2019.00026.
16. S. Shiina, S. Iwasaki, K. Taura, and P. Balaji, "Lightweight preemptive user-level threads," In Proc. of the *26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '21). Association for Computing Machinery, New York, NY, USA, 2021*, pp. 374–388. doi 10.1145/3437801.3441610.
17. J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, "An extended set of FORTRAN basic linear algebra subprograms," ACM Trans. Math. Softw., **14**, 1–17 (1988). doi 10.1145/42288.42291.
18. Intel® Math Kernel Library Documentation Library. https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-documentation.html. Cited October 21, 2025.
19. KML_BLAS Library Functions - Kunpeng BoostKit 22.0.0 Kunpeng Math Library Developer Guide 01 - Huawei. https://support.huawei.com/enterprise/en/doc/EDOC1100283141/88ccc310/kml_blas-library-functions. Cited October 21, 2025.
20. GitHub - OpenMathLib/OpenBLAS: OpenBLAS is an optimized BLAS library based on GotoBLAS2 1.13 BSD version. https://github.com/xianyi/OpenBLAS. Cited October 21, 2025.
21. R. Clint Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimization of software and the ATLAS project," Parallel Computing, **27** (1), 3–35 (2001). doi 10.1016/S0167-8191(00)00087-9.
22. Taskflow. A General-purpose Task-parallel Programming System. https://taskflow.github.io/. Cited October 21, 2025.

2025, **26** (4), 449–464. doi 10.26089/NumMet.v26r430

23. The LLVM Compiler Infrastructure Project. https://llvm.org/. Cited October 21, 2025.
24. Miniconda - Anaconda. https://www.anaconda.com/docs/getting-started/miniconda/main. Cited October 21, 2025.

Received *August 20, 2025*
Accepted *October 3, 2025*
Published *October 30, 2025*

Information about the authors

Vladimir A. Bakhtin — Ph.D., Leading Researcher; Keldysh Institute of Applied Mathematics of RAS, Miusskaya ploshchad', 4, 125047, Moscow, Russia.

Nikita A. Kataev — Researcher; Keldysh Institute of Applied Mathematics of RAS, Miusskaya ploshchad', 4, 125047, Moscow, Russia.

Alexander S. Kolganov — Ph.D., Researcher; Keldysh Institute of Applied Mathematics of RAS, Miusskaya ploshchad', 4, 125047, Moscow, Russia.

Dmitriy A. Zakharov — Junior Researcher; Keldysh Institute of Applied Mathematics of RAS, Miusskaya ploshchad', 4, 125047, Moscow, Russia.

Alexander A. Smirnov — Researcher; Keldysh Institute of Applied Mathematics of RAS, Miusskaya ploshchad', 4, 125047, Moscow, Russia.

Anton A. Malakhov — Researcher; Autonomous Non-profit Organization of Higher Education "NEIMARK University", Nartova ulitsa, 6, 603104, Nizhny Novgorod, Russia.