2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

doi 10.26089/NumMet.v27r103 УДК 004.428

Высокопроизводительная реализация экспоненциальных функций для процессоров архитектуры RISC-V

Е. А. Панова
Нижегородский государственный университет имени Н. И. Лобачевского,
Нижний Новгород, Российская Федерация
ORCID: 0000-0002-7418-0986, e-mail: elena.panova@itmm.unn.ru

В. Д. Волокитин
Нижегородский государственный университет имени Н. И. Лобачевского,
Нижний Новгород, Российская Федерация
ORCID: 0000-0003-1075-1329, e-mail: volokitin@itmm.unn.ru

Е. А. Козинов
Нижегородский государственный университет имени Н. И. Лобачевского,
Нижний Новгород, Российская Федерация
ORCID: 0000-0001-6776-0096, e-mail: evgeny.kozinov@itmm.unn.ru

И. Б. Мееров
Нижегородский государственный университет имени Н. И. Лобачевского,
Нижний Новгород, Российская Федерация
ORCID: 0000-0001-6905-2050, e-mail: meerov@vmk.unn.ru

Аннотация: Разработка высокопроизводительной векторной библиотеки стандартных математических функций является важной задачей для развития высокопроизводительных вычислений, выполняемых на платформах с архитектурой RISC-V. В данной работе приводится детальная реализация функций `exp`, `exp2` и `expm1` для форматов с плавающей точкой двойной, одинарной и половинной точности. Уделяется особое внимание аспектам точности вычислений и производительности. Выполняется подробное сравнение с другими существующими на данный момент библиотеками векторных математических функций, реализованными для архитектуры RISC-V.

Ключевые слова: экспоненциальные функции, векторные вычисления, высокопроизводительные вычисления, архитектура RISC-V.

Благодарности: При написании данной работы использовались вычислительные ресурсы суперкомпьютера ННГУ “Лобачевский”.

Для цитирования: Панова Е.А., Волокитин В.Д., Козинов Е.А., Мееров И.Б. Высокопроизводительная реализация экспоненциальных функций для процессоров архитектуры RISC-V // Вычислительные методы и программирование. 2026. **27**, № 1. 27–45. doi 10.26089/NumMet.v27r103.

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

High-performance implementation of exponential functions for RISC-V processors

Elena A. Panova
National Research Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russia
ORCID: 0000-0002-7418-0986, e-mail: elena.panova@itmm.unn.ru

Valentin D. Volokitin
National Research Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russia
ORCID: 0000-0003-1075-1329, e-mail: volokitin@itmm.unn.ru

Evgeny A. Kozinov
National Research Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russia
ORCID: 0000-0001-6776-0096, e-mail: evgeny.kozinov@itmm.unn.ru

Iosif B. Meyerov
National Research Lobachevsky State University of Nizhny Novgorod, Nizhny Novgorod, Russia
ORCID: 0000-0001-6905-2050, e-mail: meerov@vmk.unn.ru

Abstract: The development of a high-performance vector library for standard mathematical functions represents a significant objective for the advancement of HPC on platforms based on the RISC-V architecture. This paper presents a detailed implementation of the `exp`, `exp2`, and `expm1` functions for double-precision, single-precision, and half-precision floating-point types. Special attention is paid to aspects of computational accuracy and performance. A detailed comparison with other existing vector math libraries for the RISC-V architecture is provided.

Keywords: exponent functions, vector calculations, high-performance calculations, RISC-V architecture.

Acknowledgements: This work was performed using the Lobachevsky supercomputer at the Lobachevsky State University of Nizhny Novgorod.

For citation: E. A. Panova, V. D. Volokitin, E. A. Kozinov, I. B. Meyerov, "High-performance implementation of exponential functions for RISC-V processors," Numerical Methods and Programming. **27** (1), 27–45 (2026). doi 10.26089/NumMet.v27r103.

---

1. Введение. Реализации базовых математических функций востребованы при численном моделировании во многих предметных областях, включая вычислительную физику, химию, биологию, финансы. В зависимости от особенностей задачи к таким реализациям могут предъявляться различные требования как по точности, так и по производительности. Действительно, в некоторых задачах финансовой математики и компьютерной графики точность не является критическим фактором, влияющим на возможность решения задачи, тогда как время вычислений выходит на первый план. При решении задач вычислительной физики, напротив, к точности подсчета значений математических функций предъявляются повышенные требования. Понимая сложности, связанные с наличием двух противоречивых критериев (точность и время вычислений), разработчики компиляторов реализуют аппроксимации математических функций в соответствии с требованиями стандартов и дают прикладным программистам возможность влиять на детали работы алгоритма путем установки необходимых опций компилятора. Так, в компиляторах C и C++ поддерживается двойная и одинарная (в некоторых реализациях еще и половинная) точность, удовлетворяются требования GNU C по числу правильно вычисленных бит мантиссы, установленные для каждой функции, а также поддерживается набор флагов, определяющий степень толерантности к арифметическим преобразованиям, необходимость учета специальных значений и другие особенности реализации.

Кажущаяся простота реализации математических функций, так или иначе сводящаяся к вычислению некоторого полинома в точке, не должна вводить в заблуждение. Достижение требуемой точности

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

за как можно меньшее время вычислений и выполнение всех предъявляемых требований не являются элементарной задачей [1, 2]. Исторически одними из первых классов алгоритмов для вычисления математических функций были итерационные методы, в частности алгоритм Волдера [3]. Другие методы используют полиномиальную аппроксимацию с предварительной редукцией аргумента [4–7]. Подход с использованием табличных значений может быть применен в тех областях, где не требуется высокая точность, в частности для 8-битной арифметики. Современные высокопроизводительные реализации эффективно комбинируют полиномиальную аппроксимацию и табличные значения [8–10]. Тем не менее, задача разработки реализации, удовлетворяющей требованиям по точности и производительности, все еще остается актуальной [11].

Наряду с реализациями, встроенными в компиляторы языков программирования высокого уровня, распространены следующие программные библиотеки. Так, в библиотеке MPFR [12] представлены реализации математических функций с поддержкой произвольной точности. Библиотека CRLibm [13] содержит производительные реализации математических функций с корректным округлением в любой точке диапазона области определения. В библиотеке SLEEF [14] представлены векторные реализации математических функций для различных платформ. Тем не менее, при появлении новых процессорных архитектур проблема разработки библиотеки математических функций, оптимизированной для этой архитектуры, сочетающей точность вычислений с высокой производительностью, позволяющей настраивать режимы точности, и, что очень важно, позволяющей компилятору векторизовывать вычислительные циклы, вновь выходит на передний план. Именно в этом направлении и выполняется проект, некоторые из текущих результатов которого представлены в данной статье.

Проект RVVMF посвящен созданию высокопроизводительной библиотеки, реализующей стандартные функции из модуля LibM компиляторов C и C++ в виде, допускающем работу в векторизованных циклах (в терминологии Intel C++ Compiler такой модуль называется Short Vector Math Library (SVML)). Данная библиотека ориентирована на перспективные процессоры открытой и свободной от патентных отчислений архитектуры RISC-V, которая активно развивается последние 10 лет большим международным сообществом, в котором российские компании играют заметную роль. На момент написания статьи библиотека поддерживает экспоненциальные (`exp`, `exp2`, `expm1`) и логарифмические (`log`, `log2`, `log10`, `log1p`) функции, некоторые тригонометрические функции (`sin`, `cos`, `sincos`), гиперболический тангенс (`tanh`), квадратный корень (`sqrt`, `isqrt`, `hypot`), округления (`ceil`, `floor`, `trunc`, `round`, `rint`), модуль (`abs`). Часть функций находится в открытом доступе [15]. Существенной особенностью библиотеки является поддержка двойной, одинарной и половинной точности вычислений с плавающей точкой. В будущем планируется поддержка нескольких режимов точности, в частности, пониженной точности.

В данной статье представлена реализация функций `exp`, `exp2` и `expm1`. В работе приведено детальное описание алгоритма, использованных оптимизаций и улучшений для повышения точности и производительности. Выполнено подробное сравнение с другими аналогичными библиотеками под архитектуру RISC-V, а именно SLEEF и VecLibm [16, 17]. Изучен вопрос о компромиссе между скоростью вычислений и точностью, показано, какие алгоритмические приемы влияют на достижение этого компромисса. Авторы надеются, что результаты этой работы могут быть полезны другим исследователям при реализации математических библиотек. Код рассматриваемых функций находится в открытом доступе [15].

Статья является доработанной и расширенной версией публикации [18].

2. Постановка задачи. Существует ряд стандартов, описывающих требования к реализации математических функций, в частности стандарт арифметики с плавающей точкой IEEE 754, стандарт компьютерной арифметики ISO/IEC 10967, стандарты языков программирования. В них описывается набор необходимых математических функций, область определения, математические свойства, специальные случаи и особые точки, точность реализации и многое другое. Стандарты не всегда согласованы между собой. Например, в стандарте IEEE 754 требуется точная реализация математических функций с гарантированным верным округлением во всех точках, в то время как стандарт ISO/IEC 10967 позволяет допустить ошибку округления в последнем бите мантиссы. Мы дополнительно ориентировались на реализацию математической библиотеки компилятора GNU (glibc libm), которая достаточно хорошо соответствует перечисленным стандартам и учитывает особенности применения математических функций в условиях необходимости обеспечить баланс между точностью и производительностью.

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

Важным вопросом с точки зрения корректности вычислений является обработка особых случаев. Некоторые значения аргументов требуют установки флагов, соответствующих определенным исключениям при работе с плавающей точкой (*FP exceptions*): `FE_DIVBYZERO`, `FE_INEXACT`, `FE_INVALID`, `FE_OVERFLOW`, `FE_UNDERFLOW`. Часть из них устанавливается автоматически в процессе выполнения арифметических операций (например, `FE_INEXACT`), другая требует ручной обработки в зависимости от значения аргумента и применяемого алгоритма. В частности, для функции `exp` обычно представляет интерес диапазон нормализованных (*normal*) аргументов с нормализованным результатом $x \in [x_{underflow}, x_{overflow}]$, однако также необходимо учитывать область переполнения $x > x_{overflow}$, антипереполнения $x < x_{underflow}$, особые значения $x = \pm\infty$, $x = \text{NaN}$, субнормальные (*subnormal*) аргументы и результаты. Значения $x_{underflow}$ и $x_{overflow}$ зависят от типа числа с плавающей точкой. Характерные диапазоны функций `exp`, `exp2`, `expm1`, порядок их обработки и установки флагов приведены в Приложении 1.

В некоторых приложениях пользователь может гарантировать отсутствие особых значений $\pm\infty$ и NaN, что позволяет ускорить вычисления. Эти и другие "агрессивные" математические оптимизации могут быть задействованы при использовании опций компиляции, например ключа `-ffast-math` компилятора GCC. Библиотека RVVMF учитывает данную возможность и предоставляет версию функций, в которых отключены обработка исключений и проверки на особые значения.

Важным вопросом при проектировании математической библиотеки является точность реализации математических функций. Мерой точности может являться, например, максимальное значение отклонения полученного результата от истинного значения функции. Для описания ошибок округления часто используют понятие *unit in the last place* (ulp), означающее расстояние между двумя представимыми числами с плавающей точкой. Данное расстояние зависит от числа, в окрестности которого измеряется отклонение. Существует несколько определений ulp [19], незначительно различающихся между собой. В частности, в определении по Кэхэну $\text{ulp}(x)$ является расстоянием между двумя ближайшими к x представимыми числами $a \neq b$, при этом не требуется выполнение условия $a \leqslant x \leqslant b$. С другой стороны, $\text{ulp}(x)$ по Харрисону — это расстояние между ближайшими к x представимыми числами $a \neq b$, такими, что $a \leqslant x \leqslant b$, в предположении неограниченности диапазона чисел с плавающей точкой. На практике имеет смысл вычислять $\text{ulp}(x)$ по Кэхэну в окрестности точек $\pm\infty$, а в остальных случаях — по Харрисону.

Максимальное значение ошибки вычислений математической функции, не превышающее 0.5 ulp, гарантирует корректное округление во всех точках. Здесь и далее подразумевается режим округления к ближайшему представимому числу (*round-to-nearest mode*). Максимальную ошибку можно определить теоретически, не перебирая все представимые точки, посредством формальных доказательств, основанных на интервальной арифметике. Тем не менее, ограничение в 0.5 ulp достаточно трудно обеспечить, что подробно рассматривается в разделе 3.4.5. Большинство математических библиотек удовлетворяет более слабым критериям. Так, модуль libm компилятора GCC [20] гарантирует максимальную ошибку не более 1 ulp для функций `exp`, `exp2`, `expm1`. Данная формулировка дает неполное представление о точности реализации математической функции, т.к. не содержит информации о частоте случаев с неверным округлением. В связи с этим на практике для описания ошибок также применяются статистические критерии, измеряющие число неверных округлений среди случайной выборки аргументов из множества представимых точек диапазона. Если среди n случайных аргументов k значений функции было округлено неверно, и при этом максимальная ошибка между истинным и округленным значением не превышает 1 ulp, то говорят, что точность равна $0.5 + k/n$ ulp в статистическом смысле. Кроме того, представляется целесообразным определять максимальное расстояние между истинным и полученным значением функции среди точек выборки, однако в этом случае не гарантируется отсутствие точек, для которых ошибка больше найденной максимальной [21].

Библиотека RVVMF предполагает выполнение следующих ограничений на точность реализации математических функций:

- допускается ошибка 1 ulp, но не более;
- требуется точность 0.501 ulp в статистическом смысле, что эквивалентно одному неверному округлению из 1000 случайных чисел с плавающей точкой.

Производительность оптимизируется только при выполнении данных условий.

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

3. Алгоритм вычисления экспоненты.

3.1. Общий метод. Один из классических алгоритмов вычисления экспоненты включает в себя редукцию аргумента в отрезок в окрестности нуля и последующую аппроксимацию функции в этом отрезке полиномом [8, 10]. Редукция может быть выполнена за счет использования машинного представления числа с плавающей точкой: $e^x = 2^E e^y$, $y = x - E\ln 2$. При выборе $E = \lfloor x/\ln 2 \rceil$, где $\lfloor\cdot\rceil$ — округление к ближайшему целому в соответствии со стандартом IEEE, аргумент отображается в отрезок $y \in [-\ln 2/2, \ln 2/2]$, в котором функция e^y аппроксимируется достаточно точно полиномом небольшой степени. С целью уменьшения числа операций при вычислении полинома аргумент может быть отображен в меньший диапазон за счет использования предпосчитанных табличных значений:

$$e^x = 2^E 2^{2^{-k}f} e^y \approx 2^E T(f) P(y), \tag{1}$$

$$h = \left\lfloor \frac{x}{2^{-k}\ln 2} \right\rceil = 2^k E + f, \tag{2}$$

$$y = x - h \cdot 2^{-k}\ln 2. \tag{3}$$

Здесь $f = 0, \ldots, 2^k - 1$ — младшие k бит целого числа h, $2^k E$ — старшие биты h, $T(f) = 2^{2^{-k}f}$ — значение из таблицы размером 2^k, $P(y)$ — полиномиальная аппроксимация функции e^y, $y \in [-\ln 2/2^{k+1}, \ln 2/2^{k+1}]$. Значение параметра k выбирается с учетом архитектурных особенностей. Увеличение k приводит к уменьшению степени аппроксимирующего полинома и, следовательно, количеству арифметических операций, однако требует использования таблицы большего размера.

Функция e^y на редуцированном отрезке аппроксимируется полиномом. Часто используются полиномы Чебышева для получения равномерного приближения или полиномы, минимизирующие максимальную ошибку на отрезке. При построении минимаксных полиномов применяется классический алгоритм Ремеза [4] и специализированные алгоритмы, адаптированные для случая коэффициентов с плавающей точкой [5]. Для построения минимаксного полинома мы использовали программное обеспечение Sollya [22]. Ошибка аппроксимации контролируется степенью полинома. Кроме ошибок аппроксимации, на точность результата существенно влияют ошибки, возникающие при выполнении операций над числами с плавающей точкой. Поддержка *fused multiply–add* (FMA) инструкций позволяет одновременно улучшить производительность и точность вычислений, последняя из которых достигается за счет выполнения одного округления вместо двух при эквивалентном умножении и сложении. Также на некоторых этапах вычисления функции для корректного округления требуется учитывать отбрасываемые младшие биты операндов.

Особое внимание с точки зрения точности следует уделить вычислению формулы (3). В случае, когда порядок y много меньше порядка x, при вычитании двух чисел возникает катастрофическая потеря точности (*cancellation*), приводящая к неверному результату. В методе Коди–Уэйта [6, 10] константа $c = 2^{-k}\ln 2$ представляется в виде суммы двух чисел с плавающей точкой c_h и c_l, где c_h хранит $(m - d)$ старших битов мантиссы c, а c_l следующие m битов (m — длина мантиссы, d ограничивает диапазон аргументов функции, $|x| < 2^d$). Доказано, что операция $y' = x - h \cdot c_h$ выполняется абсолютно точно, а ошибка, вносимая операцией $y = y' - h \cdot c_l$, уже не является катастрофической.

Вычисление полинома реализовано с использованием схемы Горнера и операций FMA. На некоторых архитектурах может присутствовать несколько устройств FMA, в этом случае выгодно выполнять вычисления параллельно, например используя метод Эстрина [23]. Наша реализация адаптирована для одного или двух устройств FMA за счет выполнения вычислений независимо для четных и нечетных степеней полинома. Ошибки округления при вычислении полинома слабо влияют на результат, поскольку для функции e^y полином представляется в виде $P(y) = 1 + Q(y)$, где $|Q(y)| \ll 1$, следовательно, накопленная в младших битах $Q(y)$ ошибка не учитывается при сложении.

В следующих разделах представлены некоторые полезные приемы, использующиеся при вычислении экспоненты, а также несколько алгоритмов вычисления функции e^x, отличающихся соотношением точности и скорости работы.

3.2. Некоторые полезные техники. В данном разделе кратко описываются известные техники, полезные при реализации математических функций. Эти и другие приемы описаны более подробно, например, в [24, 25].

Для улучшения точности вычислений иногда необходимо сохранять и использовать младшие биты мантиссы, теряющиеся при выполнении стандартных арифметических операций. Далее операции над

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

числами, представленными двумя числами с плавающей точкой, будем называть double-FP арифметикой. Основные операции такой арифметики с доказательством корректности работы подробно описаны, например, в [24, 26]. Будем предполагать возможность использования операции FMA без выполнения промежуточного округления, что позволяет существенно сократить вычислительные затраты. В листингах 1–3 приведены некоторые функции double-FP арифметики, используемые в разделе 3.4, а именно:

- точное сложение двух чисел a и b с сохранением младших битов результата (алгоритм `fast2Sum` [25]), где порядок a не меньше порядка b (листинг 1);
- FMA с сохранением младших битов результата (листинг 2);
- умножение double-FP чисел (листинг 3).

Листинг 1. Алгоритм точного сложения с результатом double-FP (fast2Sum)

Listing 1. Accurate addition algorithm with double-FP result (fast2Sum)

```
function fast2Sum(a, b):  # exponent a >= exponent b
  rh = a + b
  rl = b - (rh - a)
  return rh, rl  # a + b = rh + rl exactly

```

Листинг 2. Алгоритм fused-multiply add с результатом double-FP

Listing 2. Fused-multiply add algorithm with double-FP result

```
function fma12(a, b, c):
  rh = FMA(a, b, c)
  rl = FMA(a, b, c - rh)
  return rh, rl

function fma12_v2(a, b, c):  # more exact
  rh = FMA(a, b, c)
  tmph, tmpl = fast2Sum(c, -rh)  # in assumption that |c| >= |rh|
  rl = FMA(a, b, tmph)
  rl += tmpl
  return rh, rl

```

Листинг 3. Алгоритм умножения двух чисел double-FP с результатом FP или double-FP

Listing 3. Algorithm for multiplying two double-FP numbers with FP or double-FP result

```
function mul22(ah, al, bh, bl):
  rh = ah * bh
  rl = FMA(ah, bh, -rh)
  rl += FMA(ah, bl, al * bh)
  return rh, rl

function mul21(ah, al, bh, bl):
  rh, rl = mul22(ah, al, bh, bl)
  return rh + rl

```

Операция округления к ближайшему целому в формуле (2) для ограниченного диапазона аргументов может быть реализована быстро за счет выполнения сдвига мантиссы и использования машинного округления. Это достигается прибавлением к исходному числу значения $1.5 \cdot 2^{m-1}$, после чего искомое целое число представляется младшими $(w-p-2)$ битами результата. Здесь m — длина мантиссы, включая старший бит, p — количество битов в порядке числа, $w = m + p$ — общее количество битов числа с плавающей точкой. Пример использования данного приема приводится в функции `calculate_h` листинга 4.

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

3.3. Базовый алгоритм вычисления функции exp. Алгоритм вычисления экспоненты (листинг 4) состоит из следующих этапов:

1. *Редукция.* Округление к ближайшему целому по формуле (2) выполняется в функции `calculate_h`, редукция аргумента по формуле (3) — в функции `range_reduction`.
2. *Извлечение табличного значения.* Младшие k бит числа h, полученного на предыдущем шаге, составляют индекс в таблице $T(f)$ размером 2^k.
3. *Вычисление полинома.* Как было отмечено в разделе 3.1, значение полинома вычисляется параллельно за счет разделения четных и нечетных степеней. Общий вид полинома и схема вычислений для полинома степени 6 с использованием метода Горнера и операций FMA продемонстрированы в формуле (4) ($1 \geqslant a_2 \geqslant a_3 \geqslant \ldots \geqslant a_6$). Единица учитывается на этапе реконструкции для обеспечения необходимой точности. Степень полинома зависит от выбора k. Функции `evaluate_polynomial_r` и `evaluate_polynomial` вычисляют соответственно значения $R(y)$ и $Q(y)$, определяемые формулой (4):

$$\begin{aligned} Q(y) = P(y) - 1 &= y + a_2y^2 + a_3y^3 + a_4y^4 + a_5y^5 + a_6y^6 \\ &= y + y^2[(a_2 + y^2(a_4 + a_6y^2))) + y(a_3 + a_5y^2))] \\ &= y + y^2[p_{even}(y^2) + y \cdot p_{odd}(y^2)] = y + y^2R(y). \end{aligned} \tag{4}$$

4. *Реконструкция.* На данном этапе выполняется реконструкция результата по формуле (1) с использованием ранее полученных значений таблицы и полинома. Для увеличения точности умножение $T(f)$ и $P(y)$ производится с помощью операции FMA: $T(f)P(y) = T(f) + T(f)Q(y)$.

Листинг 4. Базовый алгоритм вычисления функции exp
Listing 4. Base algorithm for calculating the exp function

```
function calculate_h(x):
  h = FMA(x, INV_LOG2_K, FP2INT_CONST)
  hi = AS_INT(h & MASK_HI_BIT)
  h -= FP2INT_CONST
  return h, hi

function range_reduction(x, h):
  yh_ = FMA(h, -LOG2_K_H, x)
  yh = FMA(h, -LOG2_K_L, yh_)
  return yh

function get_table_value(hi):
  fi = hi & MASK_FI_BIT  # table index
  th = TABLE_H[fi]
  return th

function evaluate_polynomial_r(yh, sqry):
  p_odd = ...   # evaluation of odd polynomial terms
  p_even = ...  # independent evaluation of even polynomial terms
  res = FMA(yh, p_odd, p_even)
  return res

function evaluate_polynomial(yh):
  sqry = yh * yh
  r = evaluate_polynomial_r(yh, sqry)
  ph = FMA(sqry, r, yh)
  return ph

function update_exponent(hi, res):   # res != 0
  Ei = hi >> k
  AS_INT(res) += Ei << (m-1)
  return res

```

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

```
34  function reconstruction(th, ph, hi):
35    res = FMA(th, ph, th)
36    res = update_exponent(hi, res)
37    return res
38
39  function exp(x):
40    # checks for overflow, underflow, NaN (omitted)
41    h, hi = calculate_h(x)
42    yh = range_reduction(x, h)
43    th = get_table_value(hi)
44    ph = evaluate_polynomial(yh)
45    res = reconstruction(th, ph, hi)
46    return res
47
```

В данном алгоритме используются следующие предвычисленные константы, параметры метода и макросы:

- `AS_INT` — получение целочисленного представления битов числа;
- `FMA` — вызов инструкции fused-multiply add;
- `INV_LOG2_K` — округленное к ближайшему представимому числу значение $\dfrac{1}{2^{-k}\ln 2}$;
- `LOG2_K_H`, `LOG2_K_L` — старшие $(m-d)$ бит и младшие m бит мантиссы числа $2^{-k}\ln 2$, подробнее см. раздел 3.1;
- `FP2INT_CONST` — число с плавающей точкой, используемое для округления к ближайшему целому и равное $1.5 \cdot 2^{m-1}$;
- `MASK_HI_BIT`, `MASK_FI_BIT` — маски для выделения младших $(p+k+1)$ и k бит соответственно;
- `TABLE_H` — таблица округленных к ближайшему представимому значений функции $T(f) = 2^{2^{-k}f}$, $f = 0, \ldots, 2^k - 1$;
- k — параметр редукции;
- m, p — длина мантиссы и порядка действующего типа числа с плавающей точкой.

В листинге 4 неявно задаются в качестве параметров степень и коэффициенты полинома $P(y)$. Степень полинома зависит от параметра k и желаемой точности аппроксимации. Для различных типов числа с плавающей точкой были выбраны следующие параметры:

- `binary64` (`double`): $k = 6$, степень полинома 6;
- `binary32` (`float`): $k = 4$, степень полинома 4;
- `binary16` (`float`): $k = 3$, степень полинома 3.

Экспериментально определено, что приведенные параметры для данного алгоритма и рассматриваемого в статье программно-аппаратного окружения (раздел 5) обеспечивают оптимальный, на наш взгляд, результат с точки зрения баланса точности и производительности.

Базовый алгоритм достаточно эффективен по производительности, однако в некоторых точках ошибка превышает 1 ulp. В следующем разделе предлагаются различные приемы, позволяющие достигнуть требуемой точности 0.501 ulp.

Стоит отметить, что при вычислении экспоненты встречаются случаи, когда результат для больших по модулю отрицательных аргументов является субнормальным (*subnormal*), но не равен нулю. Метод `update_exponent` должен учитывать особенности хранения *subnormal* и выполнять для них особую реконструкцию результата. Для краткости мы опускаем в данной статье детали работы с *subnormal* числами. Исходный код функций библиотеки [15] обеспечивает корректную обработку аргументов из всей области определения.

3.4. Вариации алгоритма вычисления функции exp. В процессе разработки библиотеки математических функций необходимо решить вопрос, касающийся баланса точности вычислений и скорости работы. Достижение правильного округления во всех точках и теоретическое обоснование корректности функции является нетривиальной задачей, которая значительно усложняется требованием обеспечения высокой производительности реализации. В частности, уточнение значения функции в достаточно ред-

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

ких “плохих” точках требует существенных дополнительных затрат, на порядок и более превосходящих ожидаемое время работы. Подробнее данный вопрос рассматривается в разделе 3.4.5.

В данном разделе приводятся пошаговые техники, позволяющие увеличить точность реализации функции `exp`. Большинство этих техник подразумевает использование double-FP арифметики. С точки зрения точности хорошим решением является выполнение всех вычислений с учетом старших и младших битов числа [25], однако это неприемлемо в рамках высокопроизводительной библиотеки математических функций. В данной работе предлагается несколько версий алгоритма вычисления экспоненты, показывающих разный результат по точности и производительности. Версии упорядочены по возрастанию точности. Каждая следующая версия заменяет операцию, приводящую в предыдущей версии к наибольшему числу ошибок округления, на ее более точный аналог. Выполнение более точных операций ухудшает производительность, что иллюстрируется в разделе 5.

§ 3.4.1. Вариация 1. Хранение старших и младших битов таблицы.

Более 95% ошибок округления для двойной точности в базовой версии алгоритма возникает из-за неточных операндов в последней операции FMA при реконструкции результата. В частности, большая потеря точности происходит из-за округления табличного значения к ближайшему представимому числу. Чтобы этого избежать, необходимо хранить младшие биты мантиссы табличного значения и учитывать их при реконструкции. Данный прием позволяет существенно увеличить точность результата без критических потерь в производительности за счет параллельного выполнения арифметических операций и операций загрузки из памяти.

В листинге 5 младшие биты таблицы хранятся в отдельном массиве `TABLE_L`. Принимается во внимание, что сложение операндов должно производиться в порядке от меньшего по модулю к большему во избежание потери точности.

Листинг 5. Вариация 1 алгоритма вычисления функции exp: хранение старших и младших битов таблицы

Listing 5. Variation 1 of the exp function calculation algorithm: storing the high and low bits of the table

```
function get_table_value(hi):
  fi = hi & MASK_FI_BIT
  th = TABLE_H[fi]
  tl = TABLE_L[fi]
  return th, tl

function reconstruction(th, tl, ph, hi):
  res = th + FMA(th, ph, tl)
  res = update_exponent(hi, res)
  return res

```

§ 3.4.2. Вариация 2. Использование double-FP арифметики при вычислении полинома.

Дальнейшее увеличение точности предполагает использование double-FP арифметики во избежание ошибок округления на этапах редукции аргумента и вычисления полинома. Известным приемом для уточнения значения полинома является точное выполнение последней операции в схеме Горнера (функция `evaluate_polynomial` листинга 6). На этапе реконструкции производится умножение двух чисел формата double-FP.

Листинг 6. Вариация 2 алгоритма вычисления функции exp: использование double-FP арифметики при вычислении полинома

Listing 6. Variation 2 of the exp function calculation algorithm: using double-FP arithmetic in polynomial calculation

```
function evaluate_polynomial(yh):
  sqry = yh * yh
  r = evaluate_polynomial_r(yh, sqry)
  ph, pl = fma12(sqry, r, yh)
  return ph, pl

```

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

```
7   function reconstruction(th, tl, ph, pl, hi):
8     sh, sl = fast2Sum(1, ph)
9     sl += pl
10    res = mul21(th, tl, sh, sl)
11    res = update_exponent(hi, res)
12    return res
13
```

§ 3.4.3. Вариация 3. Более точное выполнение редукции методом Коди–Уэйта.

В функции `range_reduction` листинга 4 выполняется редукция аргумента методом Коди–Уэйта [6]. Доказано, что первая операция FMA выполняется точно, однако вторая вносит ошибку округления. При использовании double-FP арифметики потери точности можно избежать, сохранив младшие биты y. Кроме того, существенное влияние на точность редукции в случае арифметики в одинарной точности оказывает ошибка, связанная с недостаточным количеством битов мантиссы при хранении константы $2^{-k}\ln 2$. В листинге 7 эта константа представляется в виде трех чисел с плавающей точкой, что позволяет дополнительно уточнить редуцированный аргумент.

Листинг 7. Вариация 3 алгоритма вычисления функции exp: более точное выполнение редукции методом Коди–Уэйта

Listing 7. Variation 3 of the exp function calculation algorithm: a more accurate implementation of the Cody—Waite reduction

```
1   function range_reduction(x, h):
2     yh_ = FMA(h, -LOG2_K_H, x)
3     yh, yl = fma12(h, -LOG2_K_L, yh_)
4     yl = FMA(h, -LOG2_K_LL, yl)
5     yh, yl = fast2Sum(yh, yl)
6     return yh, yl
7
8   function evaluate_polynomial(yh, yl):
9     sqry = yh * yh
10    r = evaluate_polynomial_r(yh, sqry)
11    ph, pl = fma12(sqry, r, yh)
12    pl += yl
13    return ph, pl
14
```

§ 3.4.4. Вариация 4. Использование операций FMA в double-FP арифметике.

В листинге 2 представлено несколько версий реализации операции FMA в double-FP арифметике. Ни одна из них не дает абсолютно точного результата, однако это и не требуется. Тем не менее, относительно небольшое количество ошибок округления может оказывать заметное влияние на статистику в условиях достаточно сильных требований к качеству решения. В частности, в листинге 7 замена операции `fma12` в строке 3 на более точную операцию `fma12_v2` (листинг 2) позволяет достигнуть требуемого ограничения 0.501 ulp для половинной точности во всех рассматриваемых диапазонах (раздел 5.1).

§ 3.4.5. Дальнейшее повышение точности. Обсуждение.

Полученный результат можно уточнять за счет повсеместного использования double-FP арифметики. В CRLibm [25] разработчикам таким образом удалось достигнуть точности 68 бит результата для типа `double`. Тем не менее, даже такая точность не гарантирует корректного округления. В литературе данная проблема называется *Table Maker's Dilemma (TMD)* [27, 11]. Трансцендентный результат z, найденный с некоторой точностью ε, может быть округлен как в большую, так и в меньшую сторону, если середина отрезка между двумя представимыми точками попадает в интервал $(z-\varepsilon, z+\varepsilon)$.

Классическим решением проблемы TMD является итерационный алгоритм Зива [28], предполагающий повторение вычислений в расширенной точности. Лефевр [27] показал, что для экспоненты в двойной точности достаточно 157 битов мантиссы для гарантированного выполнения верного округления. В CRLibm корректная во всех точках реализация функции `exp` включает в себя "быструю" и "точную" фа-

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

зы, причем последняя выполняется в расширенной арифметике. Необходимость выполнения второй фазы определяется некоторым условием после проведения первой фазы. “Точная” фаза на порядок медленнее “быстрой”, однако случаи, когда она требуется, достаточно редки: 1 раз примерно на 2 млн точек.

Мы не стремились получить реализацию, приближающуюся к CRLibm по точности. В то же время мы ставим себе задачу поиска компромисса между точностью и производительностью, как это принято в компиляторах языков программирования C и C++. В разделе 5 демонстрируется соотношение точности и скорости работы на примере представленных выше вариаций вычисления функции `exp`.

3.5. Функция exp2. Редукция и вычисление полинома. Функция `exp2(x)` вычисляет значение 2^x. Основание 2 позволяет упростить редукцию: операция $y = x - h \cdot 2^{-k}$ является точной в арифметике с плавающей точкой. В полиномиальном разложении появляется дополнительный коэффициент a_1 (формула 5), не равный единице:

$$Q(y) = P(y) - 1 = a_1 y + a_2 y^2 + a_3 y^3 + a_4 y^4 + \ldots = a_1 y + y^2 R(y). \tag{5}$$

В двойной и одинарной точности мы вычисляем данное выражение, используя операцию FMA в обычной или double-FP арифметике: $Q(y) = \mathrm{FMA}(a_1, y, y^2 R(y))$. В половинной точности требуется учитывать старшие и младшие биты $a_1 = a_{1h} + a_{1l}$.

3.6. Функция expm1. Реконструкция аргумента. Функция `expm1`, вычисляющая значение $e^x - 1$, имеет во многом схожую реализацию с функцией `exp`, за исключением финального вычитания единицы из полученного результата. Затруднение при реализации вызывает тот факт, что единица вносит различный вклад для различных аргументов. В окрестности нуля возвращаемое значение $P(y)$ формулы (4) гарантирует хорошую точность и отсутствие *cancellation*, в то время как для больших по модулю аргументов множитель 2^E не равен единице, что ведет к другой последовательности операций. Для скалярных реализаций часто эти случаи обрабатываются в отдельных ветках, как, например, в [29], однако это может быть неприемлемо с точки зрения производительности векторного кода. Мы используем вместо ветвлений операцию `fast2Sum` с предварительным определением числа с наибольшим порядком. Последнее может быть выполнено достаточно быстро за счет маскированных операций. Алгоритм реконструкции аргумента для функции `expm1` представлен в листинге 8.

Листинг 8. Реконструкция аргумента для функции expm1
Listing 8. Reconstruction of the argument for the expm1 function

```
function reconstruction(th, tl, ph, pl, hi):
  rh, rl = fast2Sum(1, ph)
  rl += pl
  sh, sl = mul22(th, tl, rh, rl)
  sh = update_exponent(hi, sh)
  sl = update_exponent(hi, sl)
  smax, smin = sort_numbers_by_exponent(sh, -1)
  uh, ul = fast2Sum(smax, smin)
  res = sum_3_numbers(uh, ul, sl)
  return res

```

В листинге 8 функция `sort_numbers_by_exponent` выполняет сортировку двух чисел в зависимости от их порядка. Функция `sum_3_numbers` суммирует три числа. В то время как для двойной точности достаточно просто сложить эти числа в порядке от меньшего к большему, в одинарной и половинной точности требуется выполнить более точное сложение с использованием двух операций `fast2Sum`.

Функция `expm1` весьма чувствительна к ошибкам округления в окрестности нуля. Для вычисления полинома в половинной точности мы добавили больше операций FMA в double-FP арифметике. Дополнительной сложностью работы с числами в половинной и иногда одинарной точности является тот факт, что в double-FP арифметике соответствующих формату битов порядка может не хватать для хранения младших битов мантиссы. Для решения этой проблемы в функции `expm1` использовалось представление числа в виде $z = z_h + 2^{-d} z_l$, где d — целое положительное число.

4. Векторная реализация. Реализация под архитектуру RISC-V выполнена с использованием векторных функций-интринсиков, компилирующихся в инструкции RVV 1.0. Реализация алгоритмов выпол-

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

нена в двойной, одинарной и половинной точности. Поддерживается режим `-ffast-math`, исключающий проверки на особые случаи. В будущем планируется поддержка нескольких реализаций с различной степенью точности, в том числе пониженной (3.5 ulp).

Особенностью архитектуры является возможность объединять логические векторные регистры за счет изменения параметра `LMUL`. Это может дать заметный прирост в производительности [30, 31]. Библиотека поддерживает 4 режима `LMUL` (*m1*, *m2*, *m4*, *m8*). В режиме с `LMUL=8` выполняется двойной запуск реализации с `LMUL=4` для ускорения времени работы, т.к. при `LMUL=8` количества логических регистров недостаточно для создания эффективного машинного кода.

5. Эксперименты.

5.1. Точность. В рамках проекта была создана система тестирования точности разработанной реализации. В качестве признанного эталона выбрана библиотека MPFR с точностью 200 бит, что гарантированно ведет к верному округлению результата. Тестовая система генерирует случайные числа с плавающей точкой в заданных диапазонах и вычисляет ошибку в ulp относительно эквивалентной функции MPFR. Стоит отметить, что случайное распределение не получается равномерным из-за неравномерного распределения чисел с плавающей точкой на числовой прямой. На выходе отображается статистика о количестве ошибок округления с отклонением больше 0.5 ulp, 1 ulp, 2 ulp. Согласно требованию точности 0.501 ulp, допустима не более чем одна ошибка в диапазоне от 0.5 до 1 ulp на 1000 протестированных значений, при этом ошибки более 1 ulp недопустимы. Тестирование производилось на 10^6 случайных точках во всей области определения и на 10^5 случайных точках в других диапазонах. Для половинной точности было протестировано каждое значение в диапазоне области определения.

В случае двойной точности, а также одинарной в условиях ограниченных ресурсов, для полноты картины необходимо также вычислять максимальное найденное отклонение [21]. Гарантировать полное отсутствие ошибок, превышающих 1 ulp, можно только с помощью формальной верификации на основе интервальной арифметики. Это является предметом будущей работы.

Далее мы приводим количество ошибок на 1000 случайных точек выборки для следующих характерных интервалов:

- $I_0 = (-\infty, \infty)$ — область определения функции, представляет наибольший интерес;
- $I_1 = (x_{underflow}, x_{overflow})$ — диапазон *normal* значений;
- $I_2 = (x_{underflow}, x_{underflow} + 4)$ — граница *underflow*;
- $I_3 = (x_{overflow} - 4, x_{overflow})$ — граница *overflow*;
- $I_4 = (-4, 4)$ — широкий диапазон вблизи нуля;
- $I_5 = (-\ln 2/2^{k+1}, \ln 2/2^{k+1})$ — узкий диапазон вблизи нуля, соответствующий полиномиальной аппроксимации.

Итоговая статистическая ошибка, характеризующая качество реализации, определяется в диапазоне всей области определения I_0. В табл. 1 демонстрируются результаты тестирования приведенных в разделе 3 вариаций функции `exp`. Формальным требованиям, представленным в разделе 2, удовлетворяют вариация 1 для двойной и одинарной точности и вариация 3 для половинной точности. В целях обеспечения высокой точности в области *normal* значений и других диапазонах в качестве итоговой версии выбрана вариация 2 для двойной, вариация 3 для одинарной и вариация 4 для половинной точности. В будущем планируется поддержка нескольких реализаций каждой математической функции с разной степенью точности.

Был проведен сравнительный анализ с другими библиотеками. Рассмотрены реализации из GNU C Library 2.39, модуля SVML пакета Intel oneAPI 2023, библиотек SLEEF, VecLibm. Библиотека VecLibm поддерживает только двойную точность, при этом для функций `exp` и `exp2` предоставляет несколько реализаций. Мы рассматриваем версии, соответствующие функциям `rvvlm_exp` и `rvvlm_exp2`, которые, согласно результатам тестирования, удовлетворяют ограничению 1 ulp. В данном разделе мы использовали опцию компиляции `-fno-fast-math`. Эта опция, в частности для библиотеки Intel SVML, влияет в том числе на выбор реализации функции: из нескольких доступных версий предпочтение отдается наиболее точной. В табл. 2 представлены результаты тестирования в различных диапазонах области определения. Зеленым цветом в таблице обозначены значения, соответствующие статистическому ограничению 0.501 ulp.

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

Таблица 1. Количество результатов с неверным округлением на 1000 точек выборки для различных вариаций функций `exp` в двойной (“d”), одинарной (“f”) и половинной (“h”) точности. Цвета обозначают следующие значения (в ulp): темно-зеленый (⩽0.01), светло-зеленый (⩽1), оранжевый (⩽10), светло-розовый (⩽100), темно-розовый (⩽1000). Жирным шрифтом выделена итоговая статистическая ошибка во всей области определения

Table 1. Number of results with incorrect rounding per 1000 sample points for different variations of the `exp` function in double (“d”), single (“f”), and half (“h”) precision. The colors denote the following values (in ulp): dark green (⩽0.01), light green (⩽1), orange (⩽10), light pink (⩽100), dark pink (⩽1000). The overall statistical error in the entire domain is highlighted in bold

Variations	expd				expf				exph			
	$\boldsymbol{I_0}$	I_1	I_2	I_3	$\boldsymbol{I_0}$	I_1	I_2	I_3	$\boldsymbol{I_0}$	I_1	I_2	I_3
Basic	**1.981**	3.710	242.38	244.44	**9.942**	18.360	209.98	203.80	**42.399**	73.680	183.71	161.94
1	**0.015**	0.030	1.650	1.860	**0.368**	0.580	6.780	7.090	**4.958**	8.950	28.790	41.330
2	**0.006**	0.010	0.730	0.770	**0.130**	0.230	3.360	3.870	**2.554**	4.390	30.220	41.340
3	**0.000**	0.000	0.000	0.010	**0.005**	0.000	0.240	0.350	**0.778**	1.350	16.560	21.450
4	**0.000**	0.000	0.000	0.010	**0.005**	0.000	0.050	0.070	**0.125**	0.240	0.000	0.000

Таблица 2. Количество результатов с неверным округлением на 1000 точек выборки для экспоненциальных функций различных библиотек в двойной (“d”), одинарной (“f”) и половинной (“h”) точности. Цвета обозначают следующие значения (в ulp): темно-зеленый (⩽0.01), светло-зеленый (⩽1), оранжевый (⩽10), светло-розовый (⩽100), темно-розовый (⩽1000). Жирным шрифтом выделена итоговая статистическая ошибка

Table 2. Number of results with incorrect rounding per 1000 sample points for exponential functions of various libraries in double (“d”), single (“f”), and half (“h”) precision. The colors denote the following values (in ulp): dark green (⩽0.01), light green (⩽1), orange (⩽10), light pink (⩽100), dark pink (⩽1000). The overall statistical error is highlighted in bold

	glibc	SLEEF	SVML	VecLibm	RVVMF	glibc	SLEEF	SVML	RVVMF	RVVMF
	expd					expf				exph
$\boldsymbol{I_0}$	**0.005**	**0.407**	**0.053**	**0.140**	**0.006**	**0.029**	**3.986**	**2.074**	**0.005**	**0.125**
I_1	0.020	0.800	0.100	0.170	0.010	0.110	7.490	4.090	0.000	0.240
I_2	1.040	63.110	5.360	17.660	0.730	0.610	93.240	39.660	0.240	0.000
I_3	0.810	62.950	4.360	17.530	0.770	0.720	95.820	27.550	0.350	0.000
I_4	0.010	0.410	0.110	0.120	0.020	0.040	4.770	2.850	0.010	0.240
I_5	0.000	0.000	0.000	0.000	0.000	0.020	0.140	0.150	0.000	0.210
	exp2d					exp2f				exp2h
$\boldsymbol{I_0}$	**0.005**	**0.441**	**0.069**	**0.115**	**0.008**	**0.034**	**2.649**	**3.577**	**0.009**	**0.108**
I_1	0.000	0.910	0.090	0.140	0.010	0.050	4.960	6.340	0.040	0.170
I_2	1.010	67.780	5.750	16.880	0.770	0.490	62.920	49.510	0.120	0.000
I_3	0.740	68.060	4.650	16.410	0.690	0.680	60.650	24.130	0.180	0.000
I_4	0.010	0.400	0.050	0.070	0.000	0.060	2.790	4.780	0.000	0.190
I_5	0.000	0.000	0.010	0.000	0.000	0.020	0.030	2.720	0.000	0.200
	expm1d					expm1f				expm1h
$\boldsymbol{I_0}$	**0.452**	**0.182**	**0.134**	**0.362**	**0.309**	**2.910**	**1.607**	**0.312**	**0.048**	**0.708**
I_1	0.900	0.340	0.270	0.690	0.660	5.710	3.530	0.630	0.070	1.040
I_2	0.000	171.680	0.000	0.000	0.000	0.000	104.250	0.000	0.000	0.000
I_3	100.760	0.220	1.220	95.420	0.770	101.650	1.070	8.220	0.350	0.000
I_4	0.520	0.500	0.340	0.340	0.770	4.620	3.090	0.410	0.110	1.270
I_5	0.000	0.440	0.310	0.000	0.440	0.040	3.100	0.580	0.050	1.680

Среди рассмотренных сторонних библиотек наилучшую точность в большинстве случаев ожидаемо продемонстрировал модуль libm компилятора GCC. Ограничению 0.501 ulp в диапазоне области определения в двойной точности удовлетворяют все рассмотренные библиотеки. В одинарной точности не удовлетворяют этому ограничению библиотеки SLEEF и Intel SVML для функций `exp` и `exp2` и SLEEF

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

и glibc для функции `expm1`. Много ошибок округления выявлено в областях I_2 и I_3 больших по модулю аргументов для библиотек SLEEF, SVML и VecLibm, что указывает на недостаточно точное выполнение редукции в этих интервалах.

Согласно результатам тестирования, библиотека RVVMF соответствует требованию 0.501 ulp в диапазоне области определения. Для двойной и одинарной точности данное требование также выполнено во всех рассматриваемых диапазонах. По точности наша реализация показывает результаты не хуже, чем реализация из математической библиотеки компилятора GCC. Для функции `expm1` в половинной точности возникают ошибки округления в окрестности нуля, что связано с точностью вычисления полинома. Тем не менее, формальные условия из раздела 2 выполнены. Дополнительное увеличение точности мы посчитали нецелесообразным с точки зрения производительности.

5.2. Производительность. Для замеров производительности была разработана система бенчмаркинга, измеряющая время работы математической функции в трех режимах. Один из тестов вычисляет по массиву аргументов массив значений функции (“array”), учитываются операции загрузки/выгрузки из памяти. Другой тест производит вычисления в цикле без операций загрузки/выгрузки, при этом аргумент следующего вызова зависит от результата предыдущего. Получаемое значение времени работы в тактах близко к показателю латентности функции (“latency”). Третий тест направлен на измерение показателя, эквивалентного пропускной способности (“throughput”): в цикле вычисляются 4 независимых значения математической функции. Аргументы для тестирования выбирались случайным образом из диапазона положительных *normal* значений с *normal* результатом. Результат усреднялся по примерно 25000 точкам выборки.

Эксперименты производились на процессоре Banana Pi RISC-V SpacemiT K1 (RVV 1.0, длина векторного регистра 256 бит), использовался кросс-компилятор GCC 14.2. Особенностью процессора является последовательное (in-order) выполнение инструкций, в связи с чем тесты “latency” и “throughput” демонстрируют почти одинаковое время работы. VecLibm предоставляет только интерфейс для обработки массива, в связи с чем для этой библиотеки были проведены замеры только для теста “array”. Замеры производительности выполнялись в режиме `-ffast-math`. В процессе тестирования могли возникнуть вычисления с *subnormal* операндами, например, при подсчете младших битов. Время работы измерялось с помощью инструкции `rdtime`. Экспериментально было выявлено, что единица `rdtime` для приведенной конфигурации соответствует 75 тактам процессора. Для удобства результаты замеров далее приведены в тактах.

В табл. 3 представлены замеры производительности для вариаций функции `exp` из раздела 3. Параметр `LMUL` выбран оптимальным и равен 2. Выделенные подчеркиванием значения соответствуют версии, интегрированной в библиотеку RVVMF. Данные версии демонстрируют статистическую ошибку менее 1 ulp в рассмотренных ранее диапазонах (табл. 1). Что характерно, для менее точного формата хранения числа с плавающей точкой требуется более точная реализация функции, следовательно, время обработки векторного регистра увеличивается при переходе от двойной точности к одинарной и от одинарной точности к половинной. Тем не менее, обработка одного элемента ускоряется в 1.75–1.95 раз, что является вполне приемлемым.

В табл. 4 представлено сравнение времени работы реализаций экспоненциальных функций из различных векторных библиотек под архитектуру RISC-V. Параметр `LMUL` выбран оптимальным для каждой

Таблица 3. Время в тактах обработки одного элемента векторного регистра (256 бит) для различных вариаций функций `exp` в двойной (“d”), одинарной (“f”), половинной (“h”) точности

Table 3. Time in cycles of processing one element of a vector register (256 bit) for different variations of the `exp` function in double (“d”), single (“f”), half (“h”) precision

Variations	`expd`			`expf`			`exph`		
	latency	throughput	array	latency	throughput	array	latency	throughput	array
Basic	28	26	26	13	12	12	6	5	5
1	33	31	30	15	14	14	7	7	7
2	*37*	*35*	*35*	17	16	16	9	9	9
3	43	41	41	*19*	*18*	*18*	10	10	10
4	45	42	43	20	19	19	*11*	*10*	*10*

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

Таблица 4. Время в тактах обработки одного элемента векторного регистра (256 бит) для экспоненциальных функций векторных математических библиотек для архитектуры RISC-V

Table 4. Time in cycles of processing one element of a vector register (256 bits) for exponential functions from several RISC-V vector math libraries

		`float64`			`float32`		`float16`
		VecLibm	SLEEF	RVVMF	SLEEF	RVVMF	RVVMF
`exp`	latency	—	68	37	21	19	11
	throughput	—	65	34	20	18	10
	array	32	66	35	20	18	10
`exp2`	latency	—	59	37	21	18	10
	throughput	—	57	35	20	17	9
	array	26	57	36	20	17	9
`expm1`	latency	—	114	41	51	23	14
	throughput	—	112	38	50	22	14
	array	35	112	39	50	22	14

функции. Для нашей реализации и SLEEF оптимальный `LMUL` равен 2. VecLibm предлагает значения по умолчанию 4 для `exp` и 2 для `expm1`.

Наша реализация демонстрирует более высокую производительность по сравнению библиотекой SLEEF. Функции `exp` и `exp2` для типа с плавающей точкой `float64` в 1.5–2 раза быстрее SLEEF, при этом показывают меньшее число ошибок округления. В одинарной точности выигрыш составляет около 10%, что обусловлено более высоким качеством результата (табл. 2). Реализация `expm1` быстрее SLEEF в 2–3 раза.

Библиотека VecLibm на данный момент является эталоном по скорости работы и обгоняет нашу реализацию примерно на 10%. Такая скорость может быть достигнута за счет оптимальной перестановки инструкций для последующего конвейерного выполнения. В нашем случае для рассматриваемого программно-аппаратного окружения перестановка инструкций не выполняется автоматически и требует дополнительного исследования и ручной настройки. Однако при запусках на процессоре с внеочередным (out-of-order) исполнением инструкций ситуация может кардинально измениться. Дальнейшая работа в том числе направлена на исследование скорости работы реализации на таких процессорах, при этом в связи с отсутствием соответствующего аппаратного обеспечения рассматривается возможность использования программных симуляторов.

Стоит отметить, что в будущем целесообразна оптимизация производительности отдельно под каждый параметр `LMUL`. VecLibm поддерживает только один параметр `LMUL`, фиксируемый на этапе компиляции библиотеки, что не всегда эффективно. Например, при векторизации вычислительного цикла может возникнуть ситуация, в которой использование параметра по умолчанию `LMUL=4` для функции `exp` окажется невыгодным из-за ограниченного числа регистров. Возникающая в такой ситуации нехватка регистров будет вызвана их нерациональным распределением и приведет к деградации суммарного времени работы векторизуемого цикла. В некоторых алгоритмах этот эффект может быть нивелирован за счет предвычисления большого количества экспонент и сохранения результатов в достаточно большие массивы, однако это не всегда возможно из-за особенностей алгоритма. Кроме того, использование дополнительной памяти также может привести к существенному замедлению вместо ожидаемого ускорения. Эти соображения позволяют сделать следующий вывод: в математических библиотеках для RISC-V процессоров целесообразно поддерживать оптимизированные реализации функций для всех доступных значений `LMUL`. Также имеет смысл предоставлять несколько интерфейсов одной и той же функции, например, для отдельного векторного вызова или для обработки массива элементов.

6. Заключение. Поиск компромисса между точностью реализации и производительностью для стандартных математических функций не является тривиальной задачей. Ограничение на максимальную ошибку 0.5 ulp требует вычислений в расширенной точности, что вызывает большие затруднения в случае векторной реализации. Мы ориентируемся на статистическое ограничение 0.501 ulp, однако такая точность требует дополнительных вычислительных затрат и не является необходимой для ряда приложений. Оптимальным решением в данной ситуации служит поддержка нескольких отличающихся по

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

точности версий, как это сделано, в частности, в библиотеке VecLibm. Выбор одной из версий удобно осуществлять установкой соответствующих опций компиляции.

Разработанные нами реализации функций `exp`, `exp2` и `expm1` не отстают по точности от сторонних библиотек векторных математических функций, а в ряде случаев их опережают. Точность реализации сопоставима с версией математической библиотеки компилятора GCC. С точки зрения производительности, наша реализация обеспечивает ускорение в несколько раз по сравнению с библиотекой SLEEF, но незначительно уступает библиотеке VecLibm (примерно на 10%). Преимуществом нашей реализации является поддержка двойной, одинарной и половинной точности, нескольких значений `LMUL` (1, 2, 4, 8), режима `-ffast-math`.

В дальнейшем мы планируем работу по дополнительной оптимизации производительности, в частности, адаптации под различные `LMUL`. Важным направлением работы является исследование производительности для процессоров RISC-V с внеочередным исполнением.

Приложение 1

Обработка особых аргументов и диапазонов

В табл. 5 приведен порядок обработки специальных диапазонов функций `exp`, `exp2`, `expm1` с учетом возможных флагов исключений плавающей точки. Подразумевается, что флаг `FE_INEXACT` поднимается автоматически и не требует ручной обработки.

Таблица 5. Порядок обработки особых значений и диапазонов для экспоненциальных функций

Table 5. The order of processing special values and ranges for exponential functions

<table>
<tr><th rowspan="2">Диапазон аргументов</th><th colspan="3">Функция $y = f(x)$</th></tr>
<tr><th>exp</th><th>exp2</th><th>expm1</th></tr>
<tr><td>$x < x_{underflow_0}$,
результат округляется
до $+0$ (exp, exp2) или -1 (expm1)</td><td colspan="2">$y = +0$, FE_UNDERFLOW</td><td>$y = -1$</td></tr>
<tr><td>$x \in [x_{underflow_0}, x_{underflow})$,
большие по модулю отрицательные
аргументы, результат subnormal</td><td>FE_UNDERFLOW</td><td>FE_UNDERFLOW,
если были потеряны
значимые биты</td><td>—</td></tr>
<tr><td>$x \in [x_{underflow}, -\texttt{MAX\_SUBNORMAL}) \cup$
$(\texttt{MAX\_SUBNORMAL}, x_{overflow}]$,
диапазон normal аргументов и
normal значений</td><td colspan="3">Специальная обработка не требуется</td></tr>
<tr><td>$x \in [-\texttt{MAX\_SUBNORMAL}, -0) \cup$
$(+0, \texttt{MAX\_SUBNORMAL}]$,
множество subnormal аргументов</td><td colspan="2">$y = +1$</td><td>$y = x$</td></tr>
<tr><td>$x > x_{overflow}$,
результат равен $+\infty$</td><td colspan="3">$y = +\infty$, FE_OVERFLOW</td></tr>
<tr><td>$x = +0$</td><td colspan="2" rowspan="2">$y = +1$</td><td>$y = +0$</td></tr>
<tr><td>$x = -0$</td><td>$y = -0$</td></tr>
<tr><td>$x = +\infty$</td><td colspan="3">$y = +\infty$</td></tr>
<tr><td>$x = -\infty$</td><td colspan="2">$y = +0$</td><td>$y = -1$</td></tr>
<tr><td>$x =$ qNaN (quiet NaN)</td><td colspan="3">$y =$ qNaN</td></tr>
<tr><td>$x =$ sNaN (signaling NaN)</td><td colspan="3">$y =$ qNaN, FE_INVALID</td></tr>
</table>

Список литературы

1. *Muller J.-M.* Elementary functions and approximate computing // Proceedings of the IEEE. 2020. **108**, N 12. 2136–2149. doi 10.1109/JPROC.2020.2991885.
2. *Liu W., Lombardi F., Shulte M.* A Retrospective and Prospective View of Approximate Computing [Point of View} // Proceedings of the IEEE. 2020. **108**, N 3. 394–399. doi 10.1109/JPROC.2020.2975695.
3. *Volder J.E.* The CORDIC trigonometric computing technique // IRE Transactions on electronic computers. 1959. **EC-8**, N 3. 330–334. doi 10.1109/TEC.1959.5222693.

4. *Remes E.* Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation // Comptes rendus de l'Académie des Sciences. 1934. **198**. 2063–2065.
5. *Brisebarre N., Chevillard S.* Efficient polynomial L^∞-approximations // Proc. 18th IEEE Symposium on Computer Arithmetic (ARITH '07), Jun 2007, Montpellier, IEEE, France. 2007. 169–176. doi 10.1109/ARITH.2007.17.
6. *Thacher Jr., Henry C.* Software Manual for the Elementary Functions (W. J. Cody, Jr. and W. Waite) // SIAM Review. 1982. **24**, N 1. 91–93. doi 10.1137/1024023.
7. *Kornerup P., Muller J.-M.* Extending the Range of the Cody and Waite Range Reduction Method. 2005. https://www.researchgate.net/publication/266465352_Extending_the_Range_of_the_Cody_and_Waite_Range_Reduction_Method. (Дата обращения: 21 января 2026).
8. *Tang P.-T.P.* Table-driven implementation of the exponential function in IEEE floating-point arithmetic // ACM Transactions on Mathematical Software (TOMS). 1989. **15**, N 2. 144–157. doi 10.1145/63522.214389.
9. *Tang P.-T.P.* Table-driven implementation of the logarithm function in IEEE floating-point arithmetic // ACM Transactions on Mathematical Software (TOMS). 1990. **16**, N 4. 378–400. doi 10.1145/98267.98294.
10. *Muller J.-M.* Elementary functions: Algorithms and Implementation. 3rd Edition. Birkhäuser Boston, MA, 2016. doi 10.1007/978-1-4899-7983-4.
11. *Brisebarre N., Hanrot G., Muller J.-M. et al.* Correctly-rounded evaluation of a function: why, how, and at what cost? // ACM Computing Surveys. 2025. **58**, N 1. 1–34. doi 10.1145/3747840.
12. The GNU MPFR Library. https://www.mpfr.org/. (Дата обращения: 21 января 2026).
13. *Daramy C., Defour D., de Dinechin F., Muller J.-M.* CR-LIBM: A correctly rounded elementary function library // Proc. SPIE 5205, Advanced Signal Processing Algorithms, Architectures, and Implementations XIII. SPIE, 2003. **5205**. 458–464. doi 10.1117/12.505591.
14. SLEEF Vectorized Math Library. https://sleef.org/. (Дата обращения: 21 января 2026).
15. GitHub - rvvpl/rvvmf. https://github.com/rvvpl/rvvmf. (Дата обращения: 21 января 2026).
16. *Tang P.-T.P.* An Open-Source RISC-V Vector Math Library // 2024 IEEE 31st Symposium on Computer Arithmetic (ARITH), Malaga, Spain, June 10–12, 2024. IEEE, 2024. pp. 60–67. doi 10.1109/ARITH61463.2024.00019
17. GitHub - rivosinc/veclibm: Vector math library using RISC-V vector ISA via C intrinsic. https://github.com/rivosinc/veclibm. (Дата обращения: 21 января 2026).
18. *Панова Е.А., Волокитин В.Д., Козинов Е.А., Мееров И.Б.* Высокопроизводительная реализация функций exp и expm1 для процессоров архитектуры RISC-V // Труды международной конференции "Суперкомпьютерные дни в России", 29–30 сентября 2025 г., Москва. М.: МАКС Пресс, 2025. 67–84.
19. *Muller J.M.* On the definition of ulp(x): research report / Thème SYM — Systèmes symboliques, Projet Arénaire. Rapport de recherche n° 5504, 2005. https://www.academia.edu/62418489/On_the_definition_of_ulp_x_. (Дата обращения: 21 января 2026).
20. The GNU C Library - GNU Project - Free Software Foundation (FSF). https://www.gnu.org/software/libc/manual/html_node/index.html. (Дата обращения: 21 января 2026).
21. *Gladman B., Innocente V., Mather J., Zimmermann P.* Accuracy of mathematical functions in single, double, double extended, and quadruple precision. HAL preprint. 2025. https://inria.hal.science/hal-03141101v8. (Дата обращения: 21 января 2026).
22. Sollya software tool. https://www.sollya.org/. (Дата обращения: 21 января 2026).
23. *Estrin G.* Organization of computer systems: the fixed plus variable structure computer // Papers presented at the May 3–5, 1960, western joint IRE-AIEE-ACM computer conference, May 3–5, 1960, San Francisco, California, USA. ACM, New York, NY, USA, 1960. 33–40. doi 10.1145/1460361.1460365.
24. *Dekker T.J.* A floating-point technique for extending the available precision // Numerische Mathematik. 1971. **18**, N 3. 224–242. doi 10.1007/BF01397083.
25. *Daramy C., Defour D., de Dinechin F., Muller J.-M.* CR-LIBM: The evaluation of the exponential: research report / LIP RR-2003-37, Laboratoire de l'informatique du parallélisme. HAL preprint, 2003. 2+37 p. https://hal-lara.archives-ouvertes.fr/hal-02102084/. (Дата обращения: 21 января 2026).
26. *Linnainmaa S.* Software for doubled-precision floating-point computations // ACM Transactions on Mathematical Software (TOMS). 1981. **7**, N 3. 272–283. doi 10.1145/355958.355960.
27. *Lefevre V., Muller J.-M., Tisserand A.* Toward correctly rounded transcendentals // IEEE Transactions on Computers. 1998. **47**, N 11. 1235–1243. doi 10.1109/12.736435.
28. *Ziv A.* Fast evaluation of elementary mathematical functions with correctly rounded last bit // ACM Transactions on Mathematical Software (TOMS). 1991. **17**, N 3. 410–423. doi 10.1145/114697.116813.
29. *Tang P.-T.P.* Table-driven implementation of the expm1 function in IEEE floating-point arithmetic // ACM Transactions on Mathematical Software (TOMS). 1992. **18**, N 2. 211–222. doi 10.1145/146847.146928.
30. *Volokitin V.D., Vasiliev E.P., Kozinov E.A., et al.* Improved vectorization of OpenCV algorithms for RISC-V CPUs // Lobachevskii Journal of Mathematics. 2024. **45**, N 1. 130–142. doi 10.1134/S1995080224010530.

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

31. *Pirova A., Vodeneeva A., Kovalev K., et al.* Performance optimization of BLAS algorithms with band matrices for RISC-V processors // Future Generation Computer Systems. 2025. **174**. Article No. 107936. doi 10.1016/j.future.2025.107936.

Получена *20 ноября 2025 г.*

Принята *23 декабря 2025 г.*

Опубликована *29 января 2026 г.*

Информация об авторах

Елена Анатольевна Панова — ассистент кафедры высокопроизводительных вычислений и системного программирования; Нижегородский государственный университет имени Н. И. Лобачевского, пр-кт Гагарина, д. 23, 603022, Нижний Новгород, Российская Федерация.

Валентин Дмитриевич Волокитин — старший преподаватель кафедры высокопроизводительных вычислений и системного программирования; Нижегородский государственный университет имени Н. И. Лобачевского, пр-кт Гагарина, д. 23, 603022, Нижний Новгород, Российская Федерация.

Евгений Александрович Козинов — к.т.н, доцент, доцент кафедры высокопроизводительных вычислений и системного программирования; Нижегородский государственный университет имени Н. И. Лобачевского, пр-кт Гагарина, д. 23, 603022, Нижний Новгород, Российская Федерация.

Иосиф Борисович Мееров — к.т.н, доцент, заведующий кафедрой высокопроизводительных вычислений и системного программирования; Нижегородский государственный университет имени Н. И. Лобачевского, пр-кт Гагарина, д. 23, 603022, Нижний Новгород, Российская Федерация.

References

1. J.-M. Muller, “Elementary functions and approximate computing,” Proceedings of the IEEE. **108** (12), 2136–2149 (2020). doi 10.1109/JPROC.2020.2991885.
2. W. Liu, F. Lombardi, and M. Shulte, “A Retrospective and Prospective View of Approximate Computing [Point of View},” Proceedings of the IEEE. **108** (3), 394–399 (2020). doi 10.1109/JPROC.2020.2975695.
3. J. E. Volder, “The CORDIC trigonometric computing technique,” IRE Transactions on electronic computers. **EC-8** (3), 330–334 (1959). doi 10.1109/TEC.1959.5222693.
4. E. Remes, “Sur un procédé convergent d’approximations successives pour déterminer les polynômes d’approximation,” Comptes rendus de l’Académie des Sciences. **198**, 2063–2065 (1934).
5. N. Brisebarre, S. Chevillard, “Efficient polynomial L^∞-approximations,” in *18th IEEE Symposium on Computer Arithmetic (ARITH ’07), Jun 2007, Montpellier, France* (IEEE, 2007), 169–176. doi 10.1109/ARITH.2007.17.
6. Jr. Thacher, C. Henry, “Software Manual for the Elementary Functions (W. J. Cody, Jr. and W. Waite),” SIAM Rev. **24** (1), 91–93 (1982). doi 10.1137/1024023.
7. P. Kornerup, J.-M. Muller, “Extending the Range of the Cody and Waite Range Reduction Method,” 2005, https://www.academia.edu/62418560/Extending_the_Range_of_the_Cody_and_Waite_Range_Reduction_Method. Cited January 21, 2026.
8. P.-T. P. Tang, “Table-driven implementation of the exponential function in IEEE floating-point arithmetic,” ACM Transactions on Mathematical Software (TOMS). **15** (2), 144–157 (1989). doi 10.1145/63522.214389.
9. P.-T. P. Tang, “Table-driven implementation of the logarithm function in IEEE floating-point arithmetic,” ACM Transactions on Mathematical Software (TOMS). **16** (4), 378–400 (1990). doi 10.1145/98267.98294.
10. J.-M. Muller, *Elementary functions: Algorithms and Implementation. 3rd Edition* (Birkhäuser Boston, MA, 2016). doi 10.1007/978-1-4899-7983-4.
11. N. Brisebarre, G. Hanrot, J.-M. Muller, et al., “Correctly-Rounded Evaluation of a Function: Why, How, and at What Cost?” ACM Computing Surveys. **58** (1), 1–34 (2025). doi 10.1145/3747840.
12. The GNU MPFR Library. https://www.mpfr.org/. Cited January 21, 2026.
13. C. Daramy, D. Defour, F. de Dinechin, and J.-M. Muller, “CR-LIBM: a correctly rounded elementary function library,” in *Proc. SPIE 5205, Advanced Signal Processing Algorithms, Architectures, and Implementations XIII* (SPIE, 2003), **5205**, 458–464. doi 10.1117/12.505591.
14. SLEEF Vectorized Math Library. https://sleef.org/. Cited January 21, 2026.
15. GitHub - rvvpl/rvvmf. https://github.com/rvvpl/rvvmf. Cited January 21, 2026.
16. P.-T. P. Tang, “An Open-Source RISC-V Vector Math Library,” in *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH), Malaga, Spain, June 10–12, 2024* (IEEE, 2024), pp. 60–67. doi 10.1109/ARITH61463.2024.00019.

2026, **27** (1), 27–45. doi 10.26089/NumMet.v27r103

17. GitHub - rivosinc/veclibm: Vector math library using RISC-V vector ISA via C intrinsic. https://github.com/rivosinc/veclibm. Cited January 22, 2025.
18. E. A. Panova, V. D. Volokitin, E. A. Kozinov, and I. B. Meyerov, "High-performance implementation of exp and expm1 functions for RISC-V processors," in *Proc. Int. Conf. on Russian Supercomputing Days, Moscow, Russia, September 29–30, 2025*, (MAX Press, Moscow, 2025), pp. 67–84. [in Russian].
19. J.-M. Muller, *On the definition of ulp(x): research report / Thème SYM — Systèmes symboliques, Projet Arénaire. Rapport de recherche n° 5504, 2005* Institut National de Recherche en Informatique et en Automatique (INRIA), 2005. https://www.academia.edu/62418489/On_the_definition_of_ulp_x_. Cited January 22, 2025.
20. The GNU C Library - GNU Project - Free Software Foundation (FSF). https://www.gnu.org/software/libc/manual/html_node/index.html. Cited January 22, 2025.
21. B. Gladman, V. Innocente, J. Mather, and P Zimmermann, *Accuracy of Mathematical Functions in Single, Double, Double Extended, and Quadruple Precision*, HAL preprint, 2025, https://inria.hal.science/hal-03141101v8. Cited January 22, 2025.
22. Sollya software tool. https://www.sollya.org/. Cited January 22, 2025.
23. G. Estrin, "Organization of computer systems: the fixed plus variable structure computer," in *Papers presented at the May 3–5, 1960, western joint IRE-AIEE-ACM computer conference, May 3–5, 1960, San Francisco, California, USA* (ACM, New York, NY, USA, 1960), 33–40. doi 10.1145/1460361.1460365.
24. T. J. Dekker, "A floating-point technique for extending the available precision," Numerische Mathematik. **18** (3), 224–242 (1971). doi 10.1007/BF01397083.
25. C. Daramy, D. Defour, F. de Dinechin, and J.-M. Muller, *CR-LIBM: The evaluation of the exponential: research report / LIP RR–2003–37 Laboratoire de l'informatique du parallélisme, 2003*, HAL preprint, 2+37 p. https://hal-lara.archives-ouvertes.fr/hal-02102084/. Cited January 22, 2025.
26. S. Linnainmaa, "Software for doubled-precision floating-point computations," ACM Transactions on Mathematical Software (TOMS). **7** (3), 272–283 (1981). doi 10.1145/355958.355960.
27. V. Lefevre, J.-M. Muller, and A. Tisserand, "Toward correctly rounded transcendentals," IEEE Transactions on Computers. **47** (11), 1235–1243 (1998). doi 10.1109/12.736435.
28. A. Ziv, "Fast evaluation of elementary mathematical functions with correctly rounded last bit," ACM Transactions on Mathematical Software (TOMS). **17** (3), 410–423 (1991). doi 10.1145/114697.116813.
29. P.-T. P. Tang, "Table-driven implementation of the Expm1 function in IEEE floating-point arithmetic," ACM Transactions on Mathematical Software (TOMS). **18** (2), 211–222 (1992). doi 10.1145/146847.146928.
30. V. D. Volokitin, E. P. Vasiliev, E. A. Kozinov, et al., "Improved Vectorization of OpenCV Algorithms for RISC-V CPUs," Lobachevskii J. Math. **45** (1), 130–142 (2024). doi 10.1134/S1995080224010530.
31. A. Pirova, A. Vodeneeva, K. Kovalev, et al., "Performance optimization of BLAS algorithms with band matrices for RISC-V processors," Future Generation Computer Systems. **174**, Article No. 107936 (2025). doi 10.1016/j.future.2025.107936.

Received	*Accepted*	*Published*
November 20, 2025	*December 23, 2025*	*January 29, 2026*

Information about the authors

Elena A. Panova — Assistant of the Department of High-Performance Computing and System Programming; National Research Lobachevsky State University of Nizhny Novgorod, prospekt Gagarina, 23, 603022, Nizhny Novgorod, Russia.

Valentin D. Volokitin — Senior Lecturer of the Department of High-Performance Computing and System Programming; National Research Lobachevsky State University of Nizhny Novgorod, prospekt Gagarina, 23, 603022, Nizhny Novgorod, Russia.

Evgeny A. Kozinov — PhD, Associate Professor, Associate Professor of the Department of High-Performance Computing and System Programming; National Research Lobachevsky State University of Nizhny Novgorod, prospekt Gagarina, 23, 603022, Nizhny Novgorod, Russia.

Iosif B. Meyerov — PhD, Associate Professor, Head of the Department of High-Performance Computing and System Programming; National Research Lobachevsky State University of Nizhny Novgorod, prospekt Gagarina, 23, 603022, Nizhny Novgorod, Russia.