



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. Н.И. ЛОБАЧЕВСКОГО
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ

Высокопроизводительная реализация функций `exr`, `exr2` и `exrm1` для процессоров архитектуры RISC-V

Е.А. Панова, В.Д. Волокитин, Е.А. Козинов, И.Б. Мееров

Суперкомпьютерные дни в России 2025
29-30 сентября

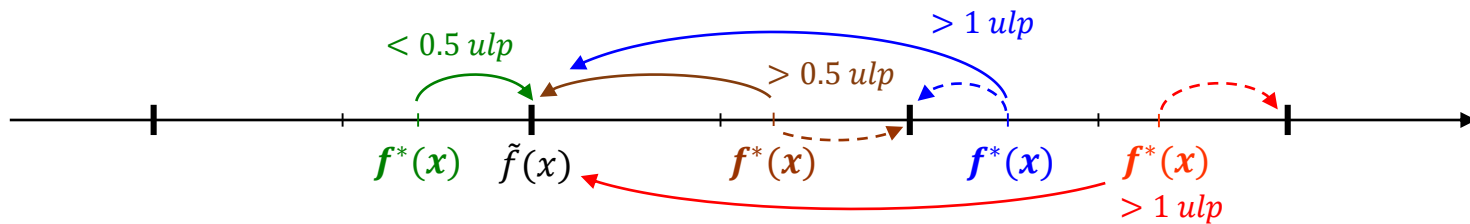
Введение

- ❑ Современные процессоры, как правило, поддерживают векторные вычисления
- ❑ Для векторизации математических вычислений необходимы *векторные математические функции*
- ❑ Производительность
- ❑ Поддержка различных режимов по точности
- ❑ Векторные библиотеки для RISC-V: SLEEF, VecLibm
- ❑ Библиотека RVVMF* (RISC-V)
 - На данный момент реализованы функции `sin`, `cos`, `sincos`, `exp`, `exp2`, `expm1`, `log`, `log2`, `log10`, `log1p`, `tanh`, `sqrt`, `hypot`, `floor`, `ceil`, `trunc`, `round`, `rint`, `abs`, `isqrt`
 - Каждая функция поддерживает три типа с плавающей точкой (`float64`, `float32`, `float16`)
- ❑ **Доклад посвящен реализации функций `exp`, `exp2`, `expm1`**

**Проект выполняется при поддержке ООО “КНС Групп”*

Unit in the last place

- Два противоречивых критерия: (1) *точность* и (2) *производительность*
- Как измерить точность? Понятие *unit in the last place (ulp)*
 - Расстояние между двумя представимыми числами с плавающей запятой
 - Ошибка вычислений может быть измерена в ulp



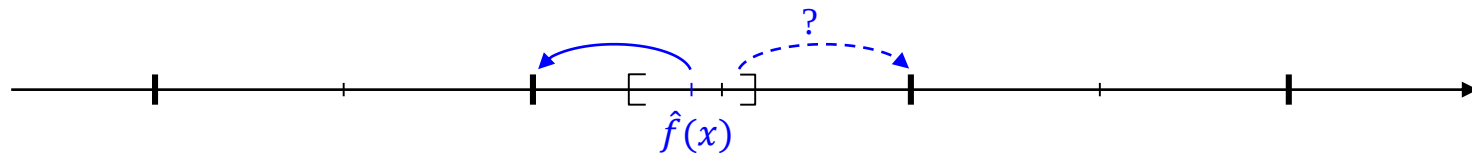
- Ошибка $\leq 1 \text{ ulp} \Rightarrow$ запрещаем «синий» и «красный» случаи
- Чем меньше случаев $> 0.5 \text{ ulp}$, тем лучше

Table Maker's Dilemma

□ Можно ли совсем исключить случаи $>0.5 \text{ ulp}$?

□ *Table Maker's Dilemma (TMD)*

- Пусть вычислено значение функции $\hat{f}(x)$ с точностью ε
- Тогда точное значение $f^*(x)$ находится в интервале $[\hat{f}(x) - \varepsilon, \hat{f}(x) + \varepsilon]$
- Если середина отрезка между двумя представимыми числами лежит в этом интервале, то куда округлять?



□ Теорема*: значение $\exp(x)$ округляется гарантированно верно, если оно вычислено с точностью 157 бит мантиссы

* C. Daramy, David Defour, Florent de Dinechin, Jean-Michel Muller. CR-LIBM: The evaluation of the exponential. 2003

Математические библиотеки

- ❑ GNU MPFR (Multiple Precision Floating-Point Reliable Library)
 - Вычисление в произвольной точности
 - Целочисленная арифметика, медленно
- ❑ CRLibm (Correctly Rounded mathematical library), CORE-MATH
 - Всегда верное округление (формальные доказательства)
 - Намного быстрее, чем MPFR (но все равно медленно)
- ❑ Glibc Libm
 - Для функций `exp`, `exp2`, `expm1` заявлена точность 1 ulp
 - Неверные округления достаточно редки
- ❑ Библиотеки векторных математических функций
 - x86-64: Intel SVML, AMD LibM, SLEEF; RISC-V: SLEEF, VecLibm
 - Как часто встречаются ошибки округления? Чему равна максимальная ошибка?*

* Gladman B. et al. Accuracy of mathematical functions in single, double, double extended, and quadruple precision. – 2025.

RVVMF. Требования

- ❑ Ошибка $\leq 1 \text{ ulp}$ для всех точек
- ❑ Точность реализации удовлетворяет ограничению 0.501 ulp
 - Допускается одно неверное округление на 1000 случайных точек
 - *Статистическое* определение ulp
- ❑ При выполнении условий выше оптимизируется производительность
- ❑ Задача условной оптимизации: поиск максимума производительности при заданных ограничениях на точность
- ❑ На текущем этапе проекта: выполнены условия на точность
- ❑ Следующие этапы проекта
 - Производительность
 - Пониженная точность (3.5 ulp) и др.

Экспонента. Простой метод

1. Редукция аргумента

$$e^x = 2^E e^y, \quad E = \left\lfloor \frac{x}{\ln 2} \right\rfloor, \quad y = x - E \ln 2, \quad y \in \left[-\frac{\ln 2}{2}, \frac{\ln 2}{2} \right]$$

2. Вычисление полинома

$$e^y = 1 + y + a_2 y + a_3 y^2 + \dots = 1 + y + y(a_2 + y(a_3 + \dots))$$

- Минимаксный полином, используем программу *sollya*
- Схема Горнера => скорость (FMA), точность
- Операция FMA – всего одно округление (важно)
- Можно вычислять на двух устройствах FMA параллельно

3. Реконструкция – восстановление результата

□ Реализации в SLEEF, VecLibm (по умолчанию)

Экспонента. Дополнительная табличная редукция*

$$e^x = 2^E \cdot 2^{2^{-k}f} \cdot e^y = 2^E \cdot T(f) \cdot P(y), \quad h = \left\lfloor \frac{x}{2^{-k} \ln 2} \right\rfloor = 2^k E + f$$

$$y = x - h \cdot 2^{-k} \ln 2, \quad y \in \left[-\frac{\ln 2}{2^{k+1}}, \frac{\ln 2}{2^{k+1}} \right]$$

- Редукция аргумента в меньший отрезок => меньше степень полинома
 - Меньше степень => меньше операций
- f – младшие k бит числа h , E – старшие биты
- $T(f) = 2^{2^{-k}f}$ – табличная функция
- k – параметр, задает размер таблицы
 - Чем больше k , тем меньше степень полинома, но больше таблица

*Tang P.T.P. Table-driven implementation of the exponential function in IEEE floating-point arithmetic //ACM Transactions on Mathematical Software (TOMS). 1989

Алгоритм. Редукция Коди-Уэйта

- $h = \lfloor z \rfloor$ легко вычислить, если прибавить и вычесть из z число $1.5 \cdot 2^{m-1}$
 - Эта операция выполняет сдвиг мантиисы, отбрасывая младшие биты и округляя
- Как вычислить $y = x - h \cdot 2^{-k} \ln 2$?
 - Возможна катастрофическая потеря точности (*cancellation*)
- Константа $c = 2^{-k} \ln 2$ разбивается особым образом на старшие и младшие биты $c = c_h + c_l$
 - c_h хранит $m - d$ битов, c_l – следующие m битов, $|x| < 2^d$, m – длина мантиисы
 - Операция $h - c_h x$ выполняется точно

```
function range_reduction(x, h):  
    yh_ = FMA(h, -ch, x)  
    yh = FMA(h, -cl, yh_)  
    return yh
```

Алгоритм. Полином. Реконструкция

- ❑ Табличное значение $t = T(f)$ – обращение к массиву размером 2^k по индексу f (младшие биты h)
- ❑ Вычисление полинома $p = P(y) - 1$
- ❑ Реконструкция: $e^x = FMA(p \cdot t + t)$
 - float64: степень полинома 5, $k=6$
 - float32: степень полинома 3, $k=4$
 - float16: степень полинома 2, $k=3$
- ❑ В дальнейшем степень увеличится на 1

```
function reconstruction(t, p, h):  
    res = FMA(t, p, t)  
    res = update_exponent(h, res)  
    return res
```

Базовый алгоритм

```
function exp(x):  
    # checks for overflow, underflow, NaN  
    h = calculate_h(x)  
    y = range_reduction(x, h)  
    t = get_table_value(h)  
    p = evaluate_polynomial(y)  
    res = reconstruction(t, p, h)  
    return res
```

- ❑ Точность: >1 ulp
- ❑ Проверки можно исключить (ключ *-ffast-math*)

Увеличение точности

- ❑ Откуда берутся ошибки?
 - Неточная аппроксимация полиномом
 - Накапливаемая *ошибка округления* при выполнении арифметических операций
- ❑ Как уменьшить ошибку округления?
 - Дополнительно хранить младшие биты
- ❑ Double-FP арифметика

Double-FP арифметика. Некоторые операции

```
function fast2Sum(a, b):  # exponent a >= exponent b
    rh = a + b
    r1 = b - (rh - a)
    return rh, r1  # a + b = rh + r1 exactly
```

```
function fma12(a, b, c):
    rh = FMA(a, b, c)
    r1 = FMA(a, b, c - rh)
    return rh, r1  # a * b + c = rh + r1
```

```
function mul22(ah, al, bh, bl):
    rh = ah * bh
    r1 = FMA(ah, bh, -rh)
    r1 += FMA(ah, bl, al * bh)
    return rh, r1  # (ah + al)*(bh + bl) = rh + r1
```

Увеличение точности. Таблица

- ❑ 95% результатов с неверным округлением в базовой версии из-за *неточного табличного значения*
- ❑ Можно хранить 2 таблицы: старшие и младшие биты
- ❑ Учитывается на этапе реконструкции
- ❑ Точность $\leq 1 \text{ ulp}$! 0.505 ulp для float64

```
function reconstruction(th, tl, p, h):  
    res = th + FMA(th, p, tl)  
    res = update_exponent(h, res)  
    return res
```

Увеличение точности. Полином

- ❑ Увеличение степени на 1 (+ одна операция FMA)
- ❑ При вычислении полинома копится *ошибка округления*
- ❑ Большой вклад в ошибку вносит неточная последняя операция FMA
- ❑ Используем операцию fma12
- ❑ Результат вычисления полинома – 2 числа, учитывается на этапе реконструкции
- ✓ Точность 0.5007 ulp для float64
- ❑ Для float32 – 0.503 ulp
 - Уточнили редукцию – 0.5001 ulp
 - Для float16 (после дополнительных уточнений) – 0.5002 ulp

Функция exp2

□ $\text{exp2}(x) = 2^x$

□ Упрощается редукция (нет *cancellation*)

$$y = x - h \cdot 2^{-k}$$

□ Усложняется полином

$$2^y = 1 + a_1 y + a_2 y^2 + a_3 y^3 + \dots$$

$$a_1 = a_{1h} + a_{1l}$$

Функция `expm1`

- ❑ $\text{expm1}(x) = e^x - 1$
- ❑ Дает более точный результат в окрестности нуля, чем $\text{exp}(x) - 1$
- ❑ Требуется точная реконструкция аргумента

Векторная реализация

- ❑ RISC-V, RVV 1.0
- ❑ Двойная, одинарная, половинная точность
- ❑ Режим *-ffast-math*
- ❑ Поддерживается LMUL=1,2,4,8
- ❑ Текущая версия: оптимальный LMUL=2

Инфраструктура

- ❑ Banana Pi RISC-V SpacemiT K1
 - RVV 1.0
 - Векторный регистр 256 бит
 - *In-order* выполнение инструкций
- ❑ Кросс-компилятор GNU 14.2

Тестирование

- ❑ Эталон по точности – MPFR (300 бит)
- ❑ Случайные числа FP в диапазоне *области определения*
 - Равновероятное FP из заданного диапазона
 - 10^6 точек
- ❑ Дополнительно протестированы диапазоны с *normal* результатом
 - $[underflow, overflow]$
 - $[-4, 4]$
 - $[underflow, underflow + 4]$ (SLEEF дает много ошибок округления для `exp` и `exp2`)
 - $[overflow - 4, overflow]$ (SLEEF дает много ошибок округления для `exp` и `exp2`)
- ❑ RVVMF в этих диапазонах: среднее число ошибок < 1 на 1000 точек

Точность

	float64				float32			float16
	glibc	SLEEF	VecLibm	<u>RVVMF</u>	glibc	SLEEF	<u>RVVMF</u>	<u>RVVMF</u>
exp	0.008	0.396	0.115	<u>0.008</u>	0.052	4.033	<u>0.016</u>	<u>0.110</u>
exp2	0.008	0.440	0.115*	<u>0.010</u>	0.047	2.603	<u>0.033</u>	<u>0.110</u>
expm1	0.476	0.187	0.392*	<u>0.326</u>	3.010	1.614	<u>0.044</u>	<u>0.693</u>

□ Среднее число ошибок на 1000 точек

□ 0.501 ulp => число в таблице меньше 1

* VecLibm – есть ошибки >1 ulp для exp2 и expm1

Производительность

	exp			exp2			expm1		
	float64	float32	float16	float64	float32	float16	float64	float32	float16
<u>RVVMF</u>	<u>32</u>	<u>17</u>	<u>9</u>	<u>30</u>	<u>15</u>	<u>8</u>	<u>39</u>	<u>22</u>	<u>14</u>
SLEEF	66	21	-	58	20	-	106	51	-
VecLibm	32	-	-	27	-	-	35	-	-

- ❑ Время в тактах на один элемент векторного регистра
- ❑ Режим *-ffast-math*

Выводы

- ❑ Реализованы векторные математические функции `exp`, `exp2`, `expm1` под архитектуру RISC-V (RVV 1.0)
- ❑ **Точность** на уровне `glibc Libm`
- ❑ **Производительность** сравнима с существующими менее точными векторными аналогами
 - SLEEF – ускорение до 2 раз
 - VecLibm – отстаем на 10%, но точнее
- ❑ **Есть реализация для `float16`**
- ❑ Открытый исходный код: <https://github.com/rvvpl/rvvmf>
- ❑ Векторная функция `exp` интегрирована в XNNPACK:
<https://github.com/google/XNNPACK/pull/8740>

Литература

- ❑ Tang P.T.P. Table-driven implementation of the exponential function in IEEE floating-point arithmetic //ACM Transactions on Mathematical Software (TOMS). 1989. V. 15. №. 2. P. 144–157. DOI: 10.1145/63522.214389.
- ❑ Cody W., Waite W. Software Manual for the Elementary Functions. Prentice-Hall, 1980. DOI: 10.1137/1024023.
- ❑ SLEEF Vectorized Math Library. URL: <https://sleef.org/>
- ❑ The VecLibm Library. URL: <https://github.com/rivosinc/veclibm>
- ❑ Tang P.T.P. An Open-Source RISC-V Vector Math Library //2024 IEEE 31st Symposium on Computer Arithmetic (ARITH). IEEE, 2024. P. 60–67.
- ❑ Dekker T.J. A floating-point technique for extending the available precision //Numerische Mathematik. 1971. V. 18. №. 3. P. 224–242. DOI: 10.1007/BF01397083
- ❑ Gladman B. et al. Accuracy of mathematical functions in single, double, double extended, and quadruple precision. – 2025.