2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

doi 10.26089/NumMet.v24r211 УДК 519.6

Быстрое вычисление показательной функции с помощью таблиц

Е. И. Васильев
Волгоградский государственный университет,
Институт математики и информационных технологий, Волгоград, Российская Федерация
ORCID: 0000-0001-9259-2696, e-mail: vasilev@volsu.ru

Г. А. Ионов
Волгоградский государственный университет,
Институт математики и информационных технологий, Волгоград, Российская Федерация
ORCID: 0000-0002-5484-1345, e-mail: pmib-201_485433@volsu.ru

М. А. Ионов
Волгоградский государственный университет,
Институт математики и информационных технологий, Волгоград, Российская Федерация
ORCID: 0000-0003-4679-9527, e-mail: pmib-201_445674@volsu.ru

Аннотация: В работе изложены алгоритмы и приведены компактные программные модули на языке C для быстрого вычисления показательной функции с помощью таблиц для процессоров архитектуры x86-64. Выполнена оценка точности и проведено сравнение быстродействия для некоторых процессоров AMD и Intel. Реализовано и протестировано обобщение табличного подхода для некоторых тригонометрических функций. В среднем предложенные функции работают в 10 раз быстрее соответствующих аналогов из стандартной математической библиотеки с прототипами в `math.h`.

Ключевые слова: трансцендентные функции, архитектура x86-64, задача Римана, ускорение вычислений, табулирование функций, AVX, AVX2, FMA.

Для цитирования: Васильев Е.И., Ионов Г.А., Ионов М.А. Быстрое вычисление показательной функции с помощью таблиц // Вычислительные методы и программирование. 2023. **24**, № 2. 142–151. doi 10.26089/NumMet.v24r211.

Fast calculation of the exponential function using tables

Eugene I. Vasilev
Volgograd State University, Institute of Mathematics and Information Technologies, Volgograd, Russia
ORCID: 0000-0001-9259-2696, e-mail: vasilev@volsu.ru

Grigorii A. Ionov
Volgograd State University, Institute of Mathematics and Information Technologies, Volgograd, Russia
ORCID: 0000-0002-5484-1345, e-mail: pmib-201_485433@volsu.ru

Mikhail A. Ionov
Volgograd State University, Institute of Mathematics and Information Technologies, Volgograd, Russia
ORCID: 0000-0003-4679-9527, e-mail: pmib-201_445674@volsu.ru

Abstract: In this paper, algorithms are described and compact software modules in the C language are presented for fast calculation of the exponential and logarithmic functions using tables for x86-64 architecture processors. The accuracy was evaluated and the performance was compared for some

2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

AMD and Intel processors. Generalization of the tabular approach has been implemented and tested for some trigonometric functions. On average, the proposed functions work 10 times faster than corresponding analogues from the standard mathematical library with prototypes in `math.h`.

Keywords: transcendental functions, architecture x86-64, Riemann problem, acceleration of calculations, tabulation of functions, AVX, AVX2, FMA.

For citation: E. I. Vasilev, G. A. Ionov, M. A. Ionov, "Fast calculation of the exponential function using tables," Numerical Methods and Programming. **24** (2), 142–151 (2023). doi 10.26089/NumMet.v24r211.

1. Введение. В общем случае показательная функция вещественного аргумента вида x^y на компьютере требует в десятки раз больше времени вычисления, чем обычные арифметические операции. Это же относится к логарифмическим и тригонометрическим функциям.

В то же время во многих областях математического моделирования аэрофизических явлений показательные функции активно используются. Уравнения кинетики химических реакций содержат эмпирические зависимости с экспонентами для скоростей реакций [1]. Для двухфазных течений и течений запыленного газа эмпирические зависимости коэффициентов сопротивления и теплообмена частиц обычно содержат степенные зависимости с рациональным показателем [2]. В методе Годунова для численного моделирования газодинамических течений используется точное решение задачи Римана (задача о распаде произвольного разрыва) [3, 4]. Задача решается итерационно, но основные затраты машинного времени (до 80%) связаны не с итерациями, а с показательными изэнтропическими функциями. В работе [5] вместо изэнтроп для волн разрежения использовали таблицы для функций двух переменных и получили существенный выигрыш по времени. В большинстве методов типа Годунова для ускорения вычислений используют приближенные процедуры для решения задачи Римана, несмотря на то что точное решение обладает рядом преимуществ [6].

В работе [7] было проведено модульное тестирование метода из работы [1] и нескольких других неявных методов при численном решении задачи взрыва кислородно-водородной смеси с целью измерения затрат времени отдельных модулей. Оказалось, что более половины времени занимает блок расчета коэффициентов скоростей химических реакций с обобщенной зависимостью Аррениуса (экспоненты). В этой же работе был предложен и реализован простой алгоритм вычисления экспоненты с помощью таблицы, который при использовании 32-разрядного компилятора mingw32-g++ сократил время работы блока скоростей химических реакций почти в 4 раза. Основной выигрыш во времени достигался за счет быстрой выборки необходимых элементов таблицы при обработке вещественного аргумента в целочисленных регистрах общего назначения.

При использовании компиляторов для 64-разрядной архитектуры эффект увеличивается, так как числа двойной точности (double) полностью помещаются в целочисленных регистрах и доступен прямой обмен данными с вещественными регистрами.

В данной работе предложены алгоритмы и программные модули для быстрого вычисления функций $\exp(x)$ и $\log_2(x)$, через которые выражается показательная функция общего вида x^y.

2. Табулирование экспоненты. Пусть вещественный x представлен в формате с фиксированной точкой в системе счисления с основанием 256 с семью цифрами после запятой (верхняя строка на рис. 1). В двоичном представлении каждая цифра занимает 1 байт, причем с учетом знака целая часть числа n_0 не превосходит 127. Такое число можно представить в виде суммы 8 элементарных слагаемых, в каждом из которых только 1 байт отличен от нуля (рис. 1). Тогда экспонента является произведением 8 элементарных экспонент.

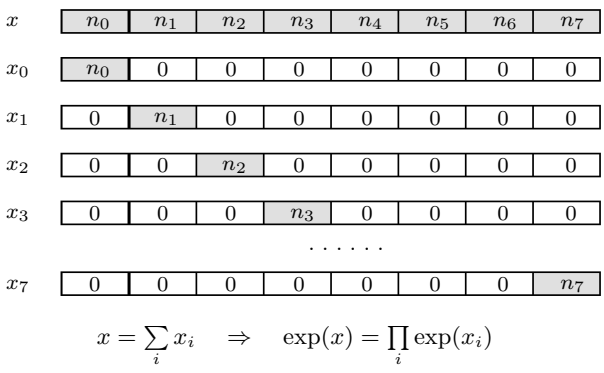

Рис. 1. Разложение $\exp(x)$ на множители для $x \in [0, 128)$

Fig. 1. Factorization of $\exp(x)$ for $x \in [0, 128)$

2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

Таблица для экспонент всех возможных восьми элементарных слагаемых будет состоять всего из $8 \times 256 = 2048$ элементов с индексацией (i, n), где i — номер цифры, n — значение цифры.

Для вычисления экспоненты нужно представить аргумент в виде 8-байтового целого, выделить содержимое отдельных байтов, извлечь сомножители из таблицы и перемножить. Всего потребуется 7 умножений. При использовании 64-битных целочисленных регистров выборку отдельных байтов можно проводить за счет быстрых побитовых операций сдвига и логического умножения.

В листинге 1 представлен код на языке C для двух функций с сопутствующими комментариями. Первая функция генерирует таблицу для положительных элементарных аргументов x_i и вызывается только один раз — в начале работы приложения. В большинстве физико-химических приложений экспонента используется с отрицательным аргументом, что учтено при формировании таблицы. Вторая функция, expT(x), использует эту таблицу и вызывается вместо библиотечной экспоненты.

Листинг 1. Код функций для быстрого вычисления $\exp(x)$

Listing 1. Code of functions for fast calculation of $\exp(x)$

```
1  #include <math.h>
2  double E2D[8][256];                        // 2D table for exp
3  double D7=256.*256*256*256*256*256*256*256; // factor 256^7
4
5  // Table Generator for exp(x)
6  void TabExp(void)
7  {  int i, n;  double x;
8     for(n=0; n<256; n++)
9     {  x=1.*n;      E2D[0][n]=exp(-x);
10       for(i=1; i<8; i++){ x=x/256.;   E2D[i][n]=exp(-x); }
11    }
12 }
13
14 // Tab-function  exp(x)
15 double expT(double x)                      // -128<x<128
16 {  int i, n, r=255;  double C, y;   long long k;
17    y=D7*fabs(x);    k=(long long)(y);  // 64-bit integer
18    n=(k>>56)&r;     C=E2D[0][n];       // left byte and Table element [0]
19    for(i=1; i<8; i++)
20    {  n=(k>>(8*(7-i)))&r;              // index of n-th element
21       C=C*E2D[i][n];                   // multiply  by n-th element
22    }
23    if(x>0.)return 1./C; else return C;
24 }
```

Сначала в строке 17 убираем знак числа, умножаем на коэффициент для переноса точки после n_7 (рис. 1) и переводим число в 8-байтовое целое. Быстрое вычисление адресов выбираемых из таблицы сомножителей реализуется в строках 18 и 20. Для положительных аргументов результат формируется проверкой и делением после цикла в строке 23.

Анализ ассемблерного кода для функции expT показал, что компилятор с опцией оптимизации `-O3` разворачивает цикл в 7 последовательных фрагментов. Нам удалось ускорить этот код на 15% за счет использования двух разных регистров при формировании адресов и загрузке для четных и нечетных сомножителей. Результаты тестирования изложены в разделе 6.

3. Табулирование логарифма. Пусть число a в нормализованной форме имеет вид $a = 2^p x$, где мантисса $1 \leqslant x < 2$. Тогда $\log_2 a = p + \log_2 x$. Будем применять кусочно-полиномиальную интерполяцию к функции $f(x) = \log_2 x$ на отрезке $[1, 2]$. Единичный отрезок разбиваем на 2^n равных отрезков. На каждом малом отрезке строим свой интерполяционный полином Эрмита 4-й степени $H_4(x)$ с заданными значениями функции и первой производной на краях и с заданным значением функции в центре. Обозначим края и середину малого отрезка как x_0, x_1, $x_s = (x_0 + x_1)/2$, тогда условиями на полином будут

$$H_4(x_0) = f(x_0), \quad H_4'(x_0) = f'(x_0), \quad H_4(x_s) = f(x_s), \quad H_4(x_1) = f(x_1), \quad H_4'(x_1) = f'(x_1). \tag{1}$$

2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

Оценим необходимое для двойной точности разбиение. Используем формулу для абсолютной погрешности интерполирования при условиях (1), например, из [8, С. 137]:

$$\Delta_4 \equiv f(x) - H_4(x) = \frac{f^{(5)}(\xi)}{5!}(x-x_0)^2(x-x_s)(x-x_1)^2, \qquad \xi \in (x_0, x_1).$$

Применительно к $f(x) = \log_2(x)$ для $x \in [1,2]$ получим

$$|\Delta_4| \leqslant \frac{1}{5\ln(2)}(x-x_0)^2|x-x_s|(x-x_1)^2.$$

Обозначим половину длины отрезка $h = (x_1 - x_0)/2 = 2^{-(n+1)}$ и $t = (x - x_s)/h$. Тогда

$$|\Delta_4| \leqslant \frac{h^5}{5\ln(2)}(t+1)^2|t|(t-1)^2 \leqslant \frac{h^5}{5\ln(2)} \max_{(-1,1)} |(t^2-1)^2 t|. \tag{2}$$

Максимум в (2) достигается при $t = 1/\sqrt{5}$. В результате получим

$$|\Delta_4| \leqslant \frac{16\sqrt{5}}{625\ln(2)}h^5 < 0.68\,\frac{h^5}{8} < \frac{1}{2^{5n+8}}.$$

При $n = 9$ абсолютная погрешность сопоставима с погрешностью формата double (53 бита). Однако для относительной погрешности этого может быть мало, так как интерполируемая функция меньше 1. В данной работе было взято $n = 10$, т.е. при интерполировании отрезок $[1,2]$ разбивался на 1024 части.

Для экономии вычислений используем схему Горнера в записи интерполяционных полиномов на каждом малом отрезке. В листинге 2 представлен код на языке C для вычисления таблицы коэффициентов

Листинг 2. Код функций для быстрого вычисления $\log_2(x)$
Listing 2. Code of functions for fast calculation of $\log_2(x)$

```
#include <math.h>
double L2D[1024][5];    //  2D table for log2

// Table Maker for log2(x). Hermite polynomial as H(x)=f0+t*(A+t*(B+t*(C+t*D)))
void TabLog2(void)
{  int i; double h=1./1024,x0,x1,f0,f1,fs,m0,m1,d,A,B,C,D,*s;
   for(i=0; i<1024; i++)
   {  x0=1.+i*h; x1=x0+h;                    // x-nodes
      f0=log2(x0); f1=log2(x1);              //  f(x)
      fs=log2(0.5*(x0+x1));      d=f1-f0;
      m0=1./x0/log(2.); m1=1./x1/log(2.);    // first derivatives
      D=16.*fs+2.*h*(m1-m0)-8.*(f1+f0);
      A=h*m0;      B=3.*d+D-h*(m1+m0+m0);
      C=h*(m1+m0)-d-d-D-D;
      s=L2D[i]; s[0]=f0; s[1]=A; s[2]=B; s[3]=C; s[4]=D;
   }
}
// Tab-function  log2(x)
double log2T(double x)              // x>0, IEEE 754-double
{  long long *u=(long long *)&x;    // u is 64-bit pointer to x
   int p=(*u>>52)-1023;             // order p selection by shift
   int n=(*u>>42)&0x3FF;            // segment index n selection by shift and &
   *u=(*u&0x3FFFFFFFFFF)|0x4090000000000000;  // removing  p and n from x
   double *s=L2D[n];                // pointer  to  n-th  row  in Table
   double t=x-1024.;                // select t coord in  n-th segment
   return  p+s[0]+t*(s[1]+t*(s[2]+t*(s[3]+t*s[4])));
}
```

2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

в этой записи для всех малых отрезков. Размер таблицы в 2.5 раза больше, чем для экспоненты. Функция построения таблицы вызывается только один раз — в начале работы приложения.

Вторая функция в листинге 2 использует таблицу для вычисления $\log_2(x)$ для произвольного $x > 0$ в формате double. Аргумент x передается в функцию через регистр xmm0. Его битовые поля в стандарте IEEE 754 изображены на рис. 2. Чтобы извлечь порядок p, нужно скопировать x в общий целочисленный регистр, сдвинуть его содержимое на 52 бита вправо и вычесть 1023. Этим действиям соответствуют строки 20 и 21 в листинге 2. Для интерполирования отрезок $[1, 2]$ разбивали на 1024 части. Десять бит мантиссы, [51:42], содержат код номера отрезка разбиения, в который попал x. Чтобы его получить, нужно сдвинуть содержимое регистра на 42 бита вправо и обнулить биты порядка. Этим действиям соответствует строка 22.

Биты [41:0] определяют относительное положение x на отрезке разбиения. Чтобы сформировать вещественное число t с дробной частью m (рис. 2), обнуляем n-биты, а на место порядка записываем 1033 (т.е. $p = 10$). Этим действиям соответствует строка 23 в листинге 2. В результате с учетом неявной единицы получим код вещественного числа $x = 2^{10}(1 + 2^{-10}t)$. После этого для получения t нужно вычесть 1024 (строка 25). Указатель в строке 24 введен для компактности записи табличных элементов. В строке 26 происходит вычисление результата по схеме Горнера. В машинных кодах такая схема быстро реализуется с помощью FMA-инструкций (Fused Multiply-Add).

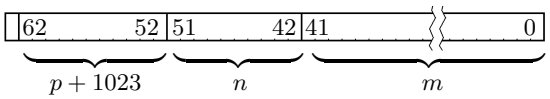

Рис. 2. Битовые поля аргумента $\log_2 x$ для $x > 0$

Fig. 2. Bit fields of argument of $\log_2 x$ for $x > 0$

Ассемблерный код компилятора для функции log2T получается достаточно коротким и эффективным, однако и его удалось ускорить на 15% за счет изменения последовательности выполнения команд.

Для реализации показательной функции $\mathrm{pow}(x, y)$ с произвольным основанием $x > 0$ используем тождество $x^y = \exp(y \cdot \ln 2 \cdot \log_2 x)$.

4. Оценка точности. Рассмотрим погрешности описанных выше табличных функций в сравнении с погрешностями аналогичных библиотечных функций для аргумента x формата double.

Для экспоненты перевод x в формат с фиксированной точкой с 56 битами после запятой (рис. 1) может приводить к потере младших разрядов мантиссы, если $|x| < 2^{-3}$. Однако это не влияет на результат. Пусть $x = a + \varepsilon$ с абсолютной погрешностью $\varepsilon < 2^{-56}$. Тогда $e^x = e^a e^\varepsilon = e^a(1 + \varepsilon + O(\varepsilon^2))$. Следовательно, относительная погрешность результата не будет превышать машинную точность 2^{-53}.

Для $\log_2(x)$ и его табличного варианта возможна высокая относительная погрешность результата при x, близких к 1, в связи с погрешностью представления исходных данных в формате double. В общем случае для оценки влияния погрешности исходных данных (при точных вычислениях) применима формула $f(x + \epsilon) = f(x) + f'(x)\epsilon + O(\epsilon^2)$. У полинома Эрмита, задаваемого условиями (1), производные на краях отрезка интерполирования совпадают с $f'(x)$. Максимальное отклонение по производной будет в центре отрезка интерполирования. Опуская несложные выкладки, приведем связь для производных:

$$H_4'(x_s) = f'(x_s) - \frac{h^4}{1920} f^{(5)}(x_s) + O(h^6). \tag{3}$$

Применительно к $f(x) = \log_2(x)$ из (3) получим

$$H_4'(x_s) = (\log_2 x_s)'(1 - \mu h^4) + O(h^6), \qquad 0 < \mu < \frac{1}{80}.$$

В нашем случае $h = 2^{-10}$. Производные отличаются очень мало. Влияние погрешности исходных данных на результат практически одинаковое.

Для более точного вычисления $\log(x)$ при близких к 1 значениях аргумента x в языке C существует библиотечная функция log1p(x)≡log(1+x). С привлечением этой функции были проведены сравнительные расчеты, которые показали, что относительная погрешность табличной функции log2T(x) сопоставима с погрешностью библиотечной функции log2(x).

Кроме погрешностей исходных данных на результат могут влиять погрешности округления в арифметических операциях в процессе вычисления функций. Для оценки этого влияния были проведены тестовые расчеты выполнимости тождества $\exp(\ln x) = x$. Погрешности результатов расчетов с применением

2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

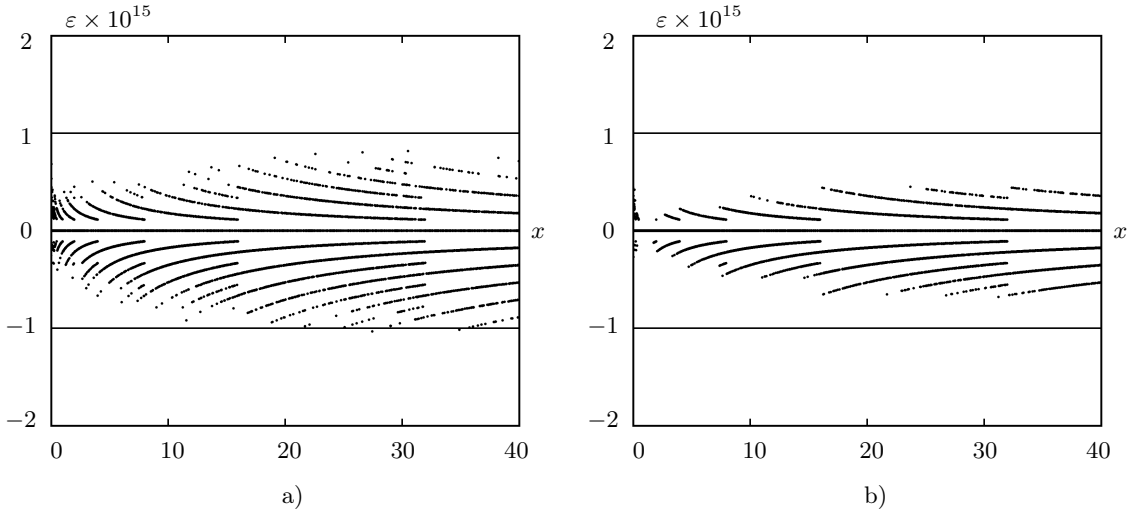

Рис. 3. Относительные погрешности вычисления величины $\exp(\ln x)/x$ для $x \in (0, 40]$:
a) табличные функции; b) функции из `math.h`

Fig. 3. Relative errors in calculating the value $\exp(\ln x)/x$ for $x \in (0, 40]$:
a) table functions; b) functions from `math.h`

табличных и идентичных им библиотечных функций представлены на рис. 3. В диапазоне $0 < x \leqslant 40$ с шагом $h = 1/256$ относительные погрешности величины $\exp(\ln x)/x$ изображены в виде дискретных точек. Такой шаг h обеспечивает отсутствие погрешности в исходных данных.

Из рис. 3 видно, что множество из 10240 расчетных точек не является непрерывной функцией. Множество распадается на несколько кластерных кривых, что характерно именно для погрешностей округления. Погрешность для табличных функций ограничена значением 10^{-15}. Для библиотечных функций погрешность приблизительно в 2 раза меньше, что, по-видимому, объясняется использованием арифметического сопроцессора (FPU) для их вычисления. В обоих случаях сравнимое количество расчетов показало нулевую погрешность: на рис. 3 a на ось x попало 22% точек, на рис. 3 b — 33%.

5. Табулирование cos(x) и sin(x). Тригонометрические функции связаны с экспонентой от мнимого аргумента формулой Эйлера: $\exp(ix) = \cos(x) + i\sin(x)$. Таблица для комплексной экспоненты отличается от таблицы для $\exp(x)$ (см. листинг 1) только удвоенным размером, так как нужно хранить вещественную и мнимую части, т.е. элементарные $\cos(x_i)$ и $\sin(x_i)$. Вычисление состоит в выборке и перемножении 8 комплексных чисел. Обычных скалярных операций умножения в 4 раза больше (28), но процедура выборки такая же, как и для $\exp(x)$. В результате получаем $\cos(x)$ и $\sin(x)$ одновременно. В следующем разделе функция на языке C, реализующая этот алгоритм, именуется как `cossinT`.

Анализ ассемблерного кода для функции `cossinT` показал, что компилятор с опцией оптимизации разворачивает цикл в 7 последовательных фрагментов, но не использует векторные возможности инструкций AVX2 для умножения комплексных чисел.

Нами был реализован векторный аналог этого алгоритма с использованием 256-битных регистров YMM. Алгоритм содержит всего 4 инструкции умножения и 3 FMA-инструкции умножения с альтернативным сложением (`vfmaddsub`) [9]. В то же время для организации параллельных вычислений пришлось применять много инструкций загрузки и перемещений внутри регистров YMM. В среднем по 3 инструкции на одну арифметическую операцию. В следующем разделе ассемблерная функция, реализующая этот векторный алгоритм, именуется как `cossinTA`.

По аналогии с расчетами, для которых погрешности вычислений представлены на рис. 3, были проведены тесты по влиянию погрешностей округления на результаты табличных тригонометрических функций. Расчеты проводились для величины $\sin^2 x + \cos^2 x$. Отличие от результатов на рис. 3 состояло лишь в том, что погрешности группировались на нескольких строго горизонтальных линиях. Модуль погрешности для `cossinT` был ограничен значением $0.5 \cdot 10^{-15}$. Для библиотечных функций максимальная погрешность оказалась в 2 раза меньше. Для табличных функций нулевую погрешность показал 51% вычислений, для библиотечных — 78% .

2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

6. Результаты тестирования. Тестирование производительности описанных выше алгоритмов и функций проводилось в свободно распространяемой оболочке Code::Blocks для 64-разрядных Windows 7 и Windows 10 с использованием компилятора `x86_64-w64-mingw32-g++.exe` с опциями `-O3` и `-march=native`. Результаты тестирования для двух пар процессоров AMD и Intel с десятилетним промежутком даты выпуска представлены в табл. 1. Частоты всех четырех процессоров лежат в диапазоне 3.3–3.8 ГГц.

В процессе тестирования с помощью каждой функции вычислялась сумма 100 млн значений на четырех непересекающихся отрезках с малым шагом. Время выполнения в миллисекундах замерялось функцией `clock()` с осреднением по шести запускам. В табл. 1 строки с номерами 1–12 разделены на 3 группы. В строках 1–4 представлены результаты для библиотечных функций (заголовочный файл `math.h`). В строках 5–8 — результаты для описанных выше табличных функций на языке C. В строках 9–12 — результаты для их аналогов на языке ассемблера. Последняя строка таблицы относится к ассемблерному варианту с инструкциями AVX2, которые процессорами A и C не поддерживаются.

Для функции `pow` задавался показатель степени $\gamma = 1.4$ (показатель адиабаты воздуха). На процессорах AMD (колонки A и B) все табличные варианты функций выполняются в 6–17 раз быстрее, чем библиотечные аналоги. Показательная функция powT в 10 с лишним раз быстрее своего аналога pow на обоих процессорах AMD, при этом скорость ее работы не зависит от показателя степени.

На процессорах Intel (колонки C и D) преимущество табличных функций еще выше, особенно для более поздней модели D. Это связано с медленной работой библиотечных функций. Измерения, отмеченные звездочкой, получены с пониженным уровнем оптимизации компилятора `-O2`. При полной оптимизации (флаг `-O3`) время выполнения в 2 раза больше. Выглядит странным, что за 10 лет скорость работы FPU, который привлекается для выполнения трансцендентных функций, изменилась незначительно. На процессорах типа D табличные функции работают в 10–19 раз быстрее библиотечных аналогов, а при использовании усиленной оптимизации это преимущество может удвоиться за счет замедления библиотечных функций. Возможно, этот парадокс связан с особенностями используемого компилятора. Для своих процессоров Intel предлагает свою версию компилятора Intel C++ Compiler.

Таблица 1. Время вычисления 100 млн значений функций (мс)

Table 1. Calculation times of 100 million function values (ms)

№	Function	Processor			
		AMD		Intel	
		FX-8150 (2011)	Ryzen 7 5700G (2021)	Core i7-3770K (2012)	Core i3-12100 (2022)
		A	B	C	D
1	$\exp(x)$	5250	1530	4370*	4140*
2	$\log2(x)$	8050	1850	1870*	1880*
3	$\mathrm{pow}(x,\gamma)$	17350	5730	7400*	6565*
4	$\cos(x)+\sin(x)$	9350	6650	6760*	6710
5	$\mathrm{expT}(x)$	770	234	580	224
6	$\mathrm{log2T}(x)$	480	173	320	141
7	$\mathrm{powT}(x,\gamma)$	1660	503	1230	473
8	$\mathrm{cossinT}(x)$	1530	598	1480	628
9	$\mathrm{expTA}(x)$	650	224	470	204
10	$\mathrm{log2TA}(x)$	420	173	310	135
11	$\mathrm{powTA}(x,\gamma)$	1370	469	1070	418
12	$\mathrm{cossinTA}(x)$	—	623	—	433

2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

Разработка ассемблерной версии табличной функции `cossinTA` с использованием инструкций AVX2 оправдала себя только для процессора D. Получено ускорение в 1.5 раза по сравнению с `cossinT` на языке C. Для той же пары функций на процессоре AMD (колонка B) эффект оказался отрицательным. Дополнительные затраты на подготовительные инструкции при использовании 256-битных регистров YMM превысили выигрыш от их использования.

После выпуска в серию AVX-процессоров в ЦЕРНе была разработана библиотека программ с открытым кодом для основных элементарных математических функций под именем VDT [10, 11]. Функции библиотеки в основном ориентированы на векторное применение. В основе большинства алгоритмов лежит дробно-рациональная аппроксимация Паде на ограниченном отрезке. Сравнение табл. 1 с данными работы [10] показывает, что наши табличные функции `expT` и `log2T` работают быстрее скалярных аналогов из библиотеки VDT примерно в 2 и 4 раза соответственно.

Заметим, что допустимый диапазон аргумента $|x| < 128$ у функции `expT` не соответствует стандарту IEEE 754–2008. Этот недостаток можно исправить за счет изменения длины элементарных слагаемых (рис. 1) с 8 до 9 бит. С учетом того, что знаковый бит не используется, получим 7 фрагментов по 9 бит. Ограничение аргумента в новом представлении будет $|x| < 512$. Недостатком такой модификации является увеличение размера таблицы с 8×256 до 7×512 элементов и уменьшение длины дробной части с 56 до 54 бит. Эксперименты с модифицированной версией функции показали, что по точности и скорости она практически не отличается от `expT`. Уменьшение количества сомножителей компенсируется более медленным вычислением номеров элементов в таблице, так как в этом случае уже не используются быстрые команды выборки байтов. Также отметим, что при использовании описанных функций в качестве программного обеспечения к ним следует добавить условия обработки исключений по допустимому значению аргумента [11], что может увеличить время работы функций на 10–15%.

7. Заключение. Для кодирования арифметических операций с вещественными числами современные 64-разрядные компиляторы вместо регистров FPU используют более быстрые регистры AVX. Однако трансцендентные команды FPU по-прежнему используются в стандартной библиотеке математических функций, что делает их низкоэффективными по сравнению с элементарными арифметическими операциями.

В данной работе описаны алгоритмы и предложены программы для быстрого вычисления некоторых математических функций с использованием таблиц. В среднем эти программы сокращают время вычисления функций в 10 раз с сохранением точности. Они будут очень полезны в тех областях численного моделирования, в которых часто используются экспонента и показательная функция: химическая кинетика, физика лазеров, неравновесная и двухфазная газодинамика.

Список литературы

1. *Васильев Е.И., Васильева Т.А.* Мульти-неявные методы с автоматическим контролем погрешности в приложениях с химическими реакциями // Журнал вычислительной математики и математической физики. 2019. **59**, № 9. 1570–1580. doi 10.1134/S0044466919090163.
2. *Васильев Е.И.* W-модификация метода С.К. Годунова и ее применение для двумерных нестационарных течений запыленного газа // Журнал вычислительной математики и математической физики. 1996. **36**, № 1. 122–135. https://www.mathnet.ru/rus/zvmmf2306.
3. *Годунов С.К., Забродин А.В., Иванов М.Я., Крайко А.Н., Прокопов Г.П.* Численное решение многомерных задач газовой динамики. М.: Наука, 1976.
4. *Куликовский А.Г., Погорелов Н.В., Семенов А.Ю.* Математические вопросы численного решения гиперболических систем уравнений. М.: Физматлит, 2001.
5. *Hornung K., Malama Yu.G., Thoma K.* Modeling of the very high velocity impact process with respect to in-situ ionization measurements // Advances in Space Research. 1996. **17**, N 12. 77–86. doi 10.1016/0273-1177(95)00762-4.
6. *Сафронов А.В.* Кинетические схемы для уравнений газодинамики // Вычислительные методы и программирование. 2009. **10**, № 1. 62–74. https://en.num-meth.ru/index.php/journal/article/view/348/355. Дата обращения: 20 марта 2023.

2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

7. *Васильев Е.И., Шатов В.А.* Модульное тестирование эффективности двух неявных методов 6-го порядка для задач химической кинетики // Труды Института математики и информационных технологий. Волгоград: Волгоградский гос. универ., 2021. 15–26. https://www.elibrary.ru/item.asp?id=48701531. Дата обращения: 20 марта 2023.
8. *Самарский А.А., Гулин А.В.* Численные методы. М.: Наука, 1989.
9. AMD64 Architecture Programmer's Manual. Volume 4: 128-bit and 256-bit Media Instructions. No 26568. November 2021. https://www.amd.com/system/files/TechDocs/26568.pdf. Дата обращения: 20 марта 2023.
10. *Hauth T., Innocente V., Piparo D.* Development and evaluation of vectorised and multi-core event reconstruction algorithms within the CMS software framework // Journal of Physics: Conference Series. 2012. **396**, Article Number 052065. doi 10.1088/1742-6596/396/5/052065.
11. CERN VDT (VectoriseD maTh) C++ Fast Math. Library. https://github.com/drbenmorgan/vdt. Дата обращения: 20 марта 2023.

Поступила в редакцию
12 февраля 2023 г.

Принята к публикации
5 марта 2023 г.

Информация об авторах

Евгений Иванович Васильев — д.ф.-м.н., профессор; Волгоградский государственный университет, Институт математики и информационных технологий, просп. Университетский, 100, 400062, Волгоград, Российская Федерация.

Григорий Александрович Ионов — студент; Волгоградский государственный университет, Институт математики и информационных технологий, просп. Университетский, 100, 400062, Волгоград, Российская Федерация.

Михаил Александрович Ионов — студент; Волгоградский государственный университет, Институт математики и информационных технологий, просп. Университетский, 100, 400062, Волгоград, Российская Федерация.

References

1. E. I. Vasilev and T. A. Vasilyeva, "Multi-Implicit Methods with Automatic Error Control in Applications with Chemical Reactions," Zh. Vychisl. Mat. Mat. Fiz. **59** (9), 1570–1580 (2019) [Comput. Math. Math. Phys. **59** (9), 1508–1517 (2019). doi 10.1134/S0965542519090161].
2. E. I. Vasilev, "A W-Modification of Godunov's Method and Its Application to Two-Dimensional Non-Stationary Flows of a Dusty Gas," Zh. Vychisl. Mat. Mat. Fiz. **36** (1), 122–135 (1996) [Comput. Math. Math. Phys. **36** (1), 101–112 (1996)]. https://dl.acm.org/doi/10.5555/229332.229355. Cited March 23, 2023.
3. S. K. Godunov, A. V. Zabrodin, M. Ya. Ivanov, A. H. Kraiko, and G. P. Prokopov, *Numerical Solution of Multidimensional Problems of Gas Dynamics* (Nauka, Moscow, 1976) [in Russian].
4. A. G. Kulikovskii, N. V. Pogorelov, and A. Yu. Semenov, *Mathematical Aspects of Numerical Solution of Hyperbolic Systems* (Fizmatlit, Moscow, 2001; CRC Press, Boca Raton, 2001).
5. K. Hornung, Yu. G. Malama, and K. Thoma, "Modeling of the Very High Velocity Impact Process with Respect to In-situ Ionization Measurements," Adv. Space Res. **17** (12), 77–86 (1996). doi 10.1016/0273-1177(95)00762-4.
6. A. V. Safronov, "Kinetic Schemes for Gas Dynamics Equations," Numerical Methods and Programming (Vychislitel'nye Metody i Programmirovanie), **10** (1), 62–74 (2009). https://en.num-meth.ru/index.php/journal/article/view/348. Cited March 23, 2023.
7. E. I. Vasilev and V. A. Shatov, "Unit Testing of the Effectiveness of Two 6th Order Implicit Methods for Chemical Kinetics Problems," in *Proc. of the Institute of Mathematics and Information Technologies* (Volgograd State University, Volgograd, 2021), pp. 15–26. https://www.elibrary.ru/item.asp?id=48701531. Cited March 23, 2023.
8. A. A. Samarskii and A. V. Gulin, *Numerical Methods* (Nauka, Moscow, 1989) [in Russian].

2023, **24** (2), 142–151. doi 10.26089/NumMet.v24r211

9. AMD64 Architecture Programmer's Manual. Volume 4: 128-bit and 256-bit Media Instructions. No 26568. November 2021. https://www.amd.com/system/files/TechDocs/26568.pdf. Cited March 20, 2023.
10. T. Hauth, V. Innocente, and D. Piparo, "Development and Evaluation of Vectorised and Multi-Core Event Reconstruction Algorithms within the CMS Software Framework," J. Phys.: Conf. Ser. **396**, Article Number 052065 (2012). doi 10.1088/1742-6596/396/5/052065.
11. CERN VDT (VectoriseD maTh) C++ Fast Math. Library. https://github.com/drbenmorgan/vdt. Cited March 20, 2023.

Received
February 12, 2023

Accepted for publication
March 5, 2023

Information about the authors

Eugene I. Vasilev — Dr. Sci., Professor; Volgograd State University, Institute of Mathematics and Information Technologies, Prosp. Universitetsky, 100, 400062, Volgograd, Russia.

Grigorii A. Ionov — Student; Volgograd State University, Institute of Mathematics and Information Technologies, Prosp. Universitetsky, 100, 400062, Volgograd, Russia.

Mikhail A. Ionov — Student; Volgograd State University, Institute of Mathematics and Information Technologies, Prosp. Universitetsky, 100, 400062, Volgograd, Russia.