计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201200135

# 基于SIMD的三角函数高性能实现与优化

姚建宇[1,2]　张祎维[3]　张广婷[1]　贾海鹏[1]
1 中国科学院计算技术研究所计算机体系结构国家重点实验室　北京 100190
2 中国科学院大学计算机与控制学院　北京 100049
3 清华大学计算机科学与技术系　北京 100084
(yaojianyu19f@ict.ac.cn)

**摘　要**　作为基本的数学运算,三角函数的高性能实现对构建处理器的基础软件生态具有重要意义,特别是当前处理器都采用了SIMD架构,基于SIMD实现高性能三角函数具有重要的研究意义和应用价值。对此,文中采用数值分析的方法,对5个常用的三角函数sin,cos,tan,atan,atan2进行了高性能的实现与优化。首先通过分析浮点数IEEE754标准,设计了高效的三角函数算法;然后通过多项式逼近算法中的泰勒公式、帕德近似及雷米兹算法提升了算法精度;最后利用指令流水线与SIMD优化进一步提升了算法性能。实验结果表明,在满足精度的前提下,所实现的三角函数,相较于libm算法库和ARM_M算法库,在ARM V8计算平台上都获得了较大的性能提升,其中相比libm算法库有1.77～6.26倍的时间性能提升,相比ARM_M算法库有1.34～1.5倍的时间性能提升。

**关键词**:三角函数;SIMD;高性能;数值分析;ARM V8架构
**中图法分类号**　TP391

## High Performance Implementation and Optimization of Trigonometric Functions Based on SIMD

YAO Jian-yu[1,2],ZHANG Yi-wei[3],ZHANG Guang-ting[1] and JIA Hai-peng[1]
1 State Key Laboratory of Computer Architecture,Institute of Computing Technology,Chinese Academy of Sciences,Beijing 100190,China
2 School of Computer and Control Engineering,University of Chinese Academy of Sciences,Beijing 100049,China
3 Department of Computer Science and Technology,Tsinghua University,Beijing 100084,China

**Abstract**　As a basic mathematical operation,the high-performance implementation of trigonometric functions is of great significance to the construction of the basic software ecology of the processor. Especially,the current processors have adopted the SIMD architecture,and the implementation of high-performance trigonometric functions based on SIMD has important research significance and application value. In this regard,this paper uses numerical analysis method to implement and optimize the five commonly used trigonometric functions sin,cos,tan,atan,atan2 with high performance. Based on the analysis of floating-point IEEE754 standard,an efficient trigonometric function algorithm is designed. Then,the algorithm accuracy is further improved by the application of Taylor formula,Pade approximation and Remez algorithm in polynomial approximation algorithm. Finally,the performance of the algorithm is further improved by using instruction pipeline and SIMD optimization. The experimental results show that,on the premise of satisfying the accuracy,the trigonometric function implemented is compared with libm algorithm library and ARM_M algorithm library,on the ARM V8 computing platform,has achieved great performance improvement,whose time performance is 1.77～6.26 times higher than libm algorithm library,and compared with ARM_M,its times performance is 1.34～1.5 times higher.

**Keywords**　Trigonometric function,SIMD,High performance,Numerical analysis,ARM V8 architecture

三角函数是基本的数学运算,也是处理器软件生态的重　　要组成部分。三角函数的应用极其宽泛,广泛应用于神经网

到稿日期:2020-12-15　返修日期:2021-04-21
基金项目:国家重点研发计划(2017YFB0202502,2018YFC0809306,2017YFB0202105,2016YFB0200803,2017YFB0202302);国家自然科学基金(61972376);北京自然科学基金(L182053)
This work was supported by the National Key R&D Program of China (2017YFB0202502, 2018YFC0809306, 2017YFB0202105, 2016YFB0200803,2017YFB0202302),National Natural Science Foundation of China(61972376) and Beijing Natural Science Foundation of China (L182053).
通信作者:张广婷(zhangguangting@ict.ac.cn)

络、数值模拟、小波变换等领域。

随着处理器架构的发展，为了更好地支持并行性，单指令多数据(Single Instruction, Multiple Data, SIMD)技术逐步发展，并已经成为当前各个处理器的主流技术，如 Intel x86 架构的 MMX/SSE/AVX 指令集、MIPS 的 MSA 指令集和 ARM 架构的 NEON 指令集[1-2]。NEON 技术是对 ARM Cortex-A 和 Cortex-R 系列处理器的 SIMD 扩展。NEON 寄存器是同一数据类型的向量。NEON 同时对多个元素运行相同的指令，支持整数和浮点数等多种数据类型的运算，还可通过加速多媒体相关算法和函数来提升多媒体应用的整体性能。

基于 SIMD 的三角函数的高性能实现与优化具有重要的研究价值和实际意义。特别是 ARM 架构的处理器市场飞速发展，其 SIMD 向量化技术也在发展[3]。因此完善 ARM SIMD 生态也尤为重要。而目前对于采用 SIMD NEON 技术的高性能三角函数实现的相关研究较少。其中一项工作是针对 SIMD 功能部件的三角函数优化算法[4]，但其研究无法赶上 ARM 架构的发展。另一项工作是对申威 26010 处理器的扩展函数库的实现与优化[5]，该工作并不涉及 ARM 处理器。因此利用 SIMD NEON 技术对 ARM 架构进行三角函数高性能实现是有必要的。三角函数的高性能实现与优化的基本思想是在保证精度的条件下，尽可能地提升算法的性能。因此，设计高精度、高性能的三角函数算法成为本文的研究重点。

因为计算机无法直接进行三角函数计算，所以三角函数的基本计算方法完全是基于数值分析[6-7]方法。其中最普遍的方法是泰勒展开，而为了优化其泰勒展开式让其在满足计算精度的同时，尽量满足计算时间，本文又使用帕德近似[8]和雷米兹算法[9-10]对泰勒展开式进行进一步优化。其中文献[8]又对帕德近似作了进一步的优化以提升精度，在满足计算精度的同时尽量提升计算时间性能；而文献[9-10]对雷米兹算法做了扩展。

我们将实现的高性能三角函数算法库与 libm 算法库[11]，ARM Performance Library(ArmPL)[12] 和 ARM_M (ARM Optimized Routines)算法库[13]进行性能比较。最终所有的函数中，单精度函数的平均误差低于 $6.00\times10^{-8}$，双精度函数的平均误差低于 $2.00\times10^{-17}$。其时间性能相比 libm 算法库提升了 1.77～6.26 倍，相比 ARM_M 算法库，提升了 1.34～1.5 倍。

本文的主要贡献如下：

(1)根据三角函数的数值分析方法，提炼了三角函数高性能实现的基本原理。

(2)提出和总结了三角函数高性能实现和优化的具体方法。

(3)实现了一个包含 sin, cos, tan, atan, atan2 在内的高性能三角函数算法库，并将其与 libm 算法库和 ARM_M 算法库进行了性能对比。

本文将讨论基于 SIMD 的三角函数高性能实现与优化。本文第 1、2 节将简述各个三角函数串行实现的基本原理，之后根据串行实现完成并行实现；第 3 节给出了并行实现的各个三角函数的性能和精度，并与已有函数库进行比较；最后总结全文并展望未来。

## 1 相关工作

本文三角函数优化的工作主要基于数值分析的相关原理利用 SIMD 进行优化，并将我们实现的高性能三角函数算法库与目前流行的三角函数算法库如 libm 算法库、ArmPL 和 ARM_M 算法库进行性能对比。

### 1.1 libm

libm 算法库是由 Red Hat 的 Corinna Vinschen 和 Jeff Johnston 进行维护的 C 库 Newlib 的一部分。Newlib 是用于嵌入式系统的一个库，这之中的 libm 是一个完整的 IEEE 数学库，该数学库提供用于嵌入式系统接收 float 和 double 两种参数的函数。

### 1.2 ARM_M

ARM_M 是由 ARM 提供的开源高性能版本的算法库。该函数库是 ARM 厂商为 Arm 处理器提供的优化的标准数学库。

## 2 三角函数实现的数值分析

计算机无法直接计算 sin, cos 等三角函数，因此要使用数值分析相关的内容。本文的三角函数主要使用泰勒展开、帕德近似、雷米兹算法来进行优化。

### 2.1 泰勒级数

如果函数 $f(x)$在点$x_0$处可导，则 $f(x)$有：

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+o(x-x_0) \tag{1}$$

当$|x-x_0|$很小时，有：

$$f(x)\approx f(x_0)+f'(x_0)(x-x_0) \tag{2}$$

等式右边是左边的近似，但是很明显右边的式子精确度不高，相比左边其误差过大。因此可将上式推广至高次多项式：

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{1}{2!}f''(x_0)(-x_0)^2+\cdots+\frac{1}{n!}f^{(n)}(x_0)(x-x_0)^n \tag{3}$$

多项式的次数越高，对应的精确度也越高，但是要达到可以满足条件的精确度，有时相应的多项式次数会过高，会造成计算时间过长，此时可以考虑使用帕德近似。

### 2.2 帕德近似

帕德近似是有理逼近的一种，给定两正整数 $m, n$ 可得 $f(x)$的$[m,n]$阶帕德近似为：

$$R(x)=\frac{\sum_{j=0}^{m}a_jx^j}{1+\sum_{k=1}^{n}b_kx^k} \tag{4}$$

实际运算中，帕德近似的精确度要比泰勒级数高很多。但是该方法只用在泰勒级数项数过高时，导致为了满足必要的精度而使计算性能大幅下降的情况。

### 2.3 雷米兹算法

建立方程 $R(x)-f(x)=Eabs(x)$，其中 $f(x)$为所要逼近的函数；$R(x)$为需要求出的逼近 $f(x)$的函数，该函数的形式为 $P(x)$或 $P(x)/Q(x)$，其中 $P(x)$和 $Q(x)$均为多项式的形式；$Eabs(x)$代表 $f(x)$和 $R(x)$的误差，最后求得的结果的值域为开始给定的范围。

$f(x)$已知，$R(x)$的项数设为想获得的多项式最高次幂 $N$，可得到方程：

$$Eabs(x)=c_0+c_1x+\cdots+c_Nx^N-f(x) \tag{5}$$

只要选取 $N+2$ 个 $x_i$ 和 $f(x_i)$，就可以建立 $N+2$ 个方程，将上式 $N+2$ 个系数求出来，但是所求的系数一般不能满足开始给定的范围。为了满足条件，要保证 $Eabs(x)$的最大最小值在给定区间内。因此，需要进行迭代，使用第一次所求的系数，找到 $Eabs(x)$的 $N+2$ 个极值点(有 $N+1$ 个根的函数有 $N+2$ 个极值点)作为新的控制点，建立 $N+2$ 个方程，求出 $N+2$ 个系数，之后判断所求的系数是否使得 $Eabs(x)$满足给定范围，若满足则停止计算，否则继续迭代，直到得出所求解。文献[7]利用该方法对 $1/x$ 进行了应用并解决一系列近似计算问题。

## 3 三角函数的实现原理

因为计算机无法直接计算三角函数，所以采用了数值分析的方法进行三角函数求解。本文一共实现了 5 个三角函数 sin，cos，tan，atan，atan2，并且均对单精度实数和双精度实数进行实现，同时每一种实现都进行了 SIMD 并行优化，下面主要介绍每个三角函数高性能实现的原理与其实现方式。

### 3.1 sin 函数与 cos 函数

sin 函数和 cos 函数分别实现了正弦和余弦的计算，这两个函数都能够实现 32 位实数单精度和 64 位实数双精度两种数据类型；由于计算原理相同，其计算过程极为相近，而 cos 与 sin 的不同之处仅在于 cos 规约过程中了添加$\frac{\pi}{2}$，即：

$$\cos(x)=\sin\left(x+\frac{\pi}{2}\right) \tag{6}$$

因此，本文仅使用 sin 算法来说明这两种算法的求解过程。

#### 3.1.1 sin 算法

sin 算法和 cos 算法的基本思想都是利用三角函数的周期性和对称性，将输入 $x$ 规约到一个小范围$\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$中，然后利用数值分析进行求解。

这里将输入 $x$ 规约到$\left[-\frac{\pi}{2},\frac{\pi}{2}\right)$范围内，该范围是保证能够并行计算的情况下，减少泰勒展开系数项的最佳范围。对于单精度的情况，将输入 $x$ 展开 5 项，即计算到$x^9$，此时展开式相比于真正的 $\sin(x)$，误差应小于：

$$\frac{(\pi/2)^{10}}{10!}\approx 2.52\times10^{-5}$$

然后使用雷米兹算法对系数进行进一步的更正优化，即可将最大相对误差缩小至 $1\times10^{-7}$ 量级。

对于双精度的情况，将输入展开 8 项，即计算到$x^{15}$，此时其误差应小于：

$$\frac{(\pi/2)^{16}}{16!}\approx 6.57\times10^{-11}$$

然后使用雷米兹算法对系数进行优化，即可将最大相对误差缩小至 $1\times10^{-16}$ 量级。

#### 3.1.2 sin 算法实现

单精度 sin 算法和双精度 sin 算法的基本思路一样，所以本文只给出单精度 sin 算法，如算法 1 所示。

**算法 1** single sin function

```
Input:x
Output:sum
1. x1=abs(x)
2. sum=x1*(1/PI)+Shift
3. odd=sum≪31
4. sum=sum-Shift
5. z=x1-sum*PI
6. z2=z*z
7. sum=(((P1*z2+P2)*z2+P3)*z2+P4)
8. *z2*z+z
9. sign=x>0? 1:-1
10. sum=sum^sign^odd
11. return sum
```

单精度 sin 算法的具体实现如下：

(1)获取输入 $x$ 的绝对值 $x_1$，如算法 1 的第 1 行所示。

(2)对 $x_1$ 的绝对值进行规约得到$\left[-\frac{\pi}{2},\frac{\pi}{2}\right)$之间的 $z$，如算法 1 的第 2－5 行所示。

首先计算 $\pi$ 整除 $x_1$ 的倍数 *sum*，之后加上 Shift($0x1.8p+23f$)实现小数部分的四舍五入。第 3 行将 *sum* 左移 31 位获得倍数的奇偶性，在第 4 行减去该值来获得最接近 $x_1$ 的整数。

(3)将规约后的 $z$ 代入泰勒展开式进行计算，如算法第 7－8 行所示。其中，$P1-P4$ 为泰勒展开式的展开项，为了满足精度，这些展开项经过了雷米兹算法的调整。

(4)恢复结果的符号位，如算法第 9－10 行所示。

(5)返回正弦值。

#### 3.1.3 sin 算法优化

sin 和 cos 函数计算过程中，采取了许多优化步骤。具体如下：

(1)算法第 1 行取绝对值操作，通过将其与 0x7fffffff 相与的位操作来获取绝对值。第 10 行取符号位操作，通过将其与 0x80000000 相与的位操作来获取符号位。为了利用位操作，32 位单精度算法使用 union 结构体 Perf_32suf 的 float f 和 unsigned int u 两个元素来灵活操作数值 $x$ 的位。而 64 位双精度算法为使用位操作，利用了 union 结构体 Perf_64suf 与其中的 double f 和 unsigned long u 两个元素。

(2)在放缩 $x$ 时，为了提高精度将算法第 5 行的 $\pi$ 拆分成 3 个数值，单精度算法下得到的数值为：

DP1_32= 0x1.921fb6p+1f

DP2_32= −0x1.777a5cp−24f

DP3_32= −0x1.ee59dap−49f

其双精度算法下得到的数值为：

DP1_64= 0x1.921fb54442d18p+1

DP2_64= 0x1.1a62633145c06p−53

DP3_64= 0x1.c1cd129024e09p−106

通过将 $x$ 依次减去 DP1，DP2，DP3，先计算精度低的位，后计算精度高的位，来保证计算的准确性。

(3)算法第 7－8 行泰勒公式的正常计算方法如下：

$$sum=P1*z^9+P2*z^7+P3*z^5+P4*z^3+z \tag{7}$$

算法 1 所应用的秦九韶方法[14]相比正常的计算方法，虽然增加了指令间的依赖，但是减少了很多计算。

(4)在使用 SIMD NEON 汇编实现高性能正弦函数时，为了进一步优化正弦函数性能，在算法 1 的第 7，8 行利用 mla 汇编指令和流水线原理计算泰勒公式，通过调整代码顺序，减少指令依赖。其计算流程如算法 2 所示。

**算法 2** SIMD NEON Taylor formula

1. temp1=mla(P2,P1,$z_2$)
2. temp2=mla(P4,P3,$z_2$)
3. sum=mla(temp2,temp1,$z_2 * z_2$)
4. sum=mla(z,sum $* z_2 *$ z)

两相邻指令计算没有依赖关系，这便于处理器利用指令流水线原理并行执行这些指令。之后的指令排布都利用了该原理。

### 3.2 tan 函数

tan 函数实现了 32 位实数单精度和 64 位实数双精度的两种算法。

#### 3.2.1 tan 算法思想

tan 算法的基本思想是利用三角函数的周期性和对称性，将输入 $x$ 规约到一个小范围中，然后利用数值分析进行求解。

这里将输入 $x$ 规约到 $\left[0,\frac{\pi}{4}\right]$。因为当输入 $x$ 趋近于 $\frac{\pi}{2}$ 时，其正切值趋近于无穷大，如果将该 $x$ 代入泰勒公式计算正切值，会因为 tan($x$)无穷大的特性增加计算机的计算误差。研究发现，当 $x$ 在区间 $\left[0,\frac{\pi}{4}\right]$ 时，其正切值在[0,1]之间，此时利用该范围的 $x$ 进行计算误差较低。该原理如图 1 所示，首先将 $x$ 规约为 $\left[0,\frac{\pi}{2}\right)$，然后 $\left[0,\frac{\pi}{2}\right)$ 可以根据与 $\frac{\pi}{4}$ 的关系分为四个区域，当 $x$ 在图 1 左右两个区间时可直接求 $x$ 的正切值，当 $x$ 在上下两个区间时求 $x+\frac{\pi}{2}$ 或 $x+\frac{3\pi}{2}$ 的正切值，之后规约为 1/tan($x$)。此时，$x$ 的取值范围为 $\left[0,\frac{\pi}{4}\right]$。

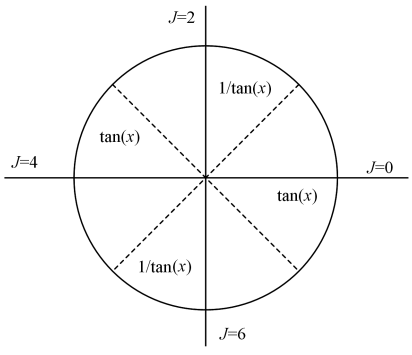


图 1 规约原理

Fig. 1 Range reduction theory

在 $x$ 规约后，对于 32 位实数单精度算法来说对 $x$ 使用泰勒展开，将其展开至 $x^{13}$，再使用雷米兹算法进一步优化系数，保证最大误差在 $1\times10^{-7}$ 量级。对于 64 位实数双精度算法来说需要将泰勒展开到 23 次方项，之后使用有理逼近的帕德近似对系数进行进一步更正优化，最后的最大误差缩小至 $1\times10^{-16}$ 量级。

#### 3.2.2 tan 算法实现

单精度 tan 算法和双精度 tan 算法的基本思路一样，所以本文只给出单精度 tan 算法，如算法 3 所示。

**算法 3** single tan function

Input:x

Output:sum

1. $x_1$=abs(x)
2. sum=$x_1$ * (2/PI)+Shift
3. odd=sum≪31
4. sum=sum−Shift
5. z=$x_1$−sum * PI/4
6. $z_2$=z * z
7. sum=(((((P1 * $z_2$+P2) * $z_2$+P3) * $z_2$+P4) * $z_2$+P5) * $z_2$+P6) * $z_2$ * z+z
8. if odd then
9. sum=1.0/sum
10. end if
11. sign=x>0? 1:−1
12. sum=sum^sign^odd
13. return sum

单精度 tan 算法的具体实现如下。

(1)获取输入 $x$ 的绝对值，如算法 3 的第 1 行所示。

(2)对输入 $x_1$ 进行规约，如算法 3 的第 2−5 行所示。首先计算输入 $x$ 的 $\frac{\pi}{2}$ 的倍数，之后加上 Shift($0x1.8p+23f$)实现四舍五入，第 3 行将 *sum* 左移 31 位获得倍数的奇偶性，在第 4 行减去该值来获得最接近 $x_1$ 的整数。

(3)将规约后的 z 代入泰勒展开式进行计算，如算法的第 7−8 行所示。其中，$P1-P6$ 为泰勒展开式的展开项，为了满足精度，这些展开项经过了雷米兹算法的调整。

(4)求得|$x$|的正切值，如算法第 9−11 行所示。如图 1 所示，通过 $x$ 的 $\frac{\pi}{2}$ 的倍数的奇偶性判断是否将 $x$ 规约为 1/tan($x$)，如果 odd 为 0x80000000，则规约为 1/tan($x$)，反之不规约。

(5)恢复结果的符号位，如算法第 12−13 行所示。

(6)返回正切值。

#### 3.2.3 tan 算法优化

tan 算法的计算过程中，采取了很多优化步骤。

(1)算法第 1 行取绝对值操作，通过将其与 0x7fffffff 相与的位操作来获取绝对值。第 12 行取符号位操作，通过将其与 0x80000000 相与的位操作来获取符号位。为了利用位操作，32 位单精度算法使用 union 结构体 Perf_32suf 的 float f 和 unsigned int u 两个元素来灵活操作数值 $x$ 的位。而 64 位双精度算法为了使用位操作，利用了 union 结构体 Perf_64suf 与其中 double f 和 unsigned long u 两个元素。

(2)在放缩 $x$ 时，为了提高精度将算法 3 第 5 行的 $\frac{\pi}{4}$ 拆分成 3 个数值，单精度算法下得到的数值为：

DP1_32=1.5703125

DP2_32=$4.837512969970703125\times10^{-4}$

DP3_32=$7.549789548918816\times10^{-8}$

其双精度算法下得到的数值为：

DP1_64＝1.570 796 310 901 641 845 703 12

DP2_64＝1.589 325 471 229 585 673 428×$10^{-8}$

DP3_64＝6.123 233 995 736 765 886 14×$^{-17}$

(3)算法 3 中第 7－8 行泰勒公式的计算方法如下：

$$sum=P1*z^{13}+P2*z^{11}+P3*z^{9}+P4*z^{7}+P5*z^{5}+P6*z^{3}+z \tag{8}$$

算法 3 所描述的秦九韶方法[14]相比一般的计算方法，虽然增加了指令间的依赖，但是减少了很多计算。

在使用 SIMD NEON 汇编实现高性能正切函数时，为了进一步优化 tan 性能，在算法 3 第 11－12 行计算泰勒公式或帕德近似的地方利用 mla 汇编指令和流水线原理，通过调整代码顺序，减少指令依赖。其算法原理与算法 2 相同。

### 3.3 atan 函数与 atan2 函数

atan2 和 atan 函数实现了相同的功能：反正切。只是 atan 的输入为一个参数，即角度；而 atan2 的输入为横纵坐标值 $x$，$y$，用 $x/y$ 即可获得角度。因此，本文使用 atan 函数来说明这两种算法的求解过程。

#### 3.3.1 atan 算法思想

atan 算法的基本思想就是利用三角函数的周期性和对称性，将输入 $x$ 规约到一个小范围 $\left(-\frac{\pi}{8},\frac{\pi}{8}\right)$ 中，然后利用数值分析进行求解。

研究发现，将角度 $x$ 利用公式

$$\tan(\alpha\pm\beta)=\frac{\tan\alpha\pm\tan\beta}{1\mp\tan\alpha\cdot\tan\beta} \tag{9}$$

缩小到 $x=\tan(t)$，$t\in\left(-\frac{\pi}{8},\frac{\pi}{8}\right)$，$x\in(-0.414\,213,0.414\,213)$ 时，$x$ 在泰勒展开第 13 次方项上有 $\frac{1}{13}x^{13}\in(-8.13\times10^{-7},8.13\times10^{-7})$。因为 32 位实数的反正切函数的精度要求为 $1\times10^{-6}$，可以看到第 13 次方项及更高次方项对精度没有帮助，因此没有计算的必要。所以在 $x$ 缩放到 $x=\tan(t)$，$t\in\left(-\frac{\pi}{8},\frac{\pi}{8}\right)$，其中 $x\in(-0.414\,213,0.414\,213)$ 的情况下，泰勒公式只需要展开到 11 次方即可，这样可以在满足计算精度的同时，尽量减少计算时间。而对双精度算法来说，$x$ 在泰勒展开第 29 次方项上时该 $x$ 项低于需要的精度，因此只展开到 27 次方项。该泰勒展开式虽然满足精度，但在计算时间上表现过差。因此本文使用了帕德近似对其进行更正优化。

规约的原理为：设 $t=\arctan(x)$，$x=\tan(t)$。

(1)当 $0\leqslant t\leqslant\frac{\pi}{8}$ 时，$t=\arctan(x)$。

(2)当 $\frac{\pi}{8}\leqslant t\leqslant\frac{3\pi}{8}$ 时，$t=\frac{\pi}{4}+\arctan\frac{x+1}{x-1}$。

(3)当 $\frac{3\pi}{8}\leqslant x\leqslant\frac{\pi}{2}$ 时，$t=\frac{\pi}{2}+\arctan\left(-\frac{1}{x}\right)$。

规约后对于 32 位实数单精度算法来说，对 $x$ 使用泰勒展开，将其展开至 $x^{11}$，再使用雷米兹算法进一步优化系数，保证最大误差在 $1\times10^{-7}$ 量级。对于 64 位双精度算法来说，需要将 $x$ 泰勒展开到 23 次方项，之后使用有理逼近的帕德近似对系数进行进一步更正优化，最终的最大误差缩小至 $1\times10^{-16}$ 量级。

#### 3.3.2 atan 算法实现

单精度 atan 算法和双精度 atan 算法的基本思路一样，所以本文只给出单精度 atan 算法，如算法 4 所示。

**算法 4** single atan function

```
Input:x
Output:sum
1. x1=abs(x)
2. if x1>tan(3PI/8) then
3.    x1=-1/x1
4.    sum=PI/2
5. else if x2>tan(PI/8) then
6.    x1=(x1-1)/(x1+1)
7.    sum=PI/4
8. end if
9. x2=x1*x1
10. sum=sum+x1+(((P1*x2+P2)*x2+P3)*x2+P4) *x2*x1
11. sign=x>0? 1:-1
12. sum=sum^sign
13. return sum
```

其具体实现如下所示：

(1)获取输入 $x$ 的绝对值，如算法 4 的第 1 行所示。

(2)对输入 $x_1$ 进行规约，如算法 4 的第 2－8 行所示。

当 $x_1$ 大于 tan(3PI/8)时，如 3.3.1 节规约原理(3)所述，计算相应的 $x_1$ 和 $sum$；当 $x_1$ 大于 tan(PI/8)时，如 3.3.1 节规约原理(2)所述，计算相应的 $x_1$ 和 $sum$。

(3)将规约后的 $x_1$ 代入泰勒展开式进行计算，如算法第 10－11 行所示。其中的 $P1-P4$ 为泰勒展开式的展开项，为了满足精度，这些展开项经过了雷米兹算法的调整。

(4)恢复结果的符号位，如算法第 12－13 行所示。

(5)返回反正切值。

#### 3.3.3 atan 算法的优化

为了获得更好的性能，atan 算法采取了许多优化步骤。

(1)算法 4 第 1 行取绝对值操作，通过将其和 0x7fffffff 相与的位操作来获取绝对值。第 12 行取符号位操作，通过将其和 0x80000000 相与的位操作来获取符号位。为了利用位操作，32 位单精度算法使用 union 结构体 Perf_32suf 的 float f 和 unsigned int u 两个元素来灵活操作数值 $x$ 的位。而 64 位双精度算法为了使用位操作，利用了 union 结构体 Perf_64suf 与其中 double f 和 unsigned long u 两个元素。

(2)算法第 10－11 行泰勒公式的一般计算方法如下：

$$sum=sum+P1*x^{9}+P2*x^{7}+P3*x^{5}+P4*x^{3}+x_1 \tag{10}$$

算法 4 所描述的秦九韶方法相比一般的计算方法，虽然增加了指令间的依赖，但是减少了很多计算。

(3)在使用 SIMD NEON 汇编实现高性能反正切函数时，为了进一步优化性能，在算法 4 第 2－8 行规约 $x$ 时，减少除法计算，其计算流程如算法 5 所示。

**算法 5** SIMD NEON $x$

```
1. larger1=cgt(x1,tan(PI/8))
```

2. larger2=cgt($x_1$, tan(3PI/8))
3. sum=bsl(lager1, PI/4, sum)
4. temp1=bsl(larger1, $x_1$−1, 1)
5. temp2=bsl(larger2, PI/2, sum)
6. sum=bsl(larger2, $x_1$, temp1)
7. temp1=bsl(larger2, $x_1$, temp1)
8. temp2=bsl(larger2, −1, temp2)
9. $x_1$=div(temp2, temp1)
10. sum=add(sum, $x_1$)

如算法 5 所示，相较于直接将算法 4 的第 2−8 行翻译成 SIMD 指令，算法 5 会减少一个除法运算。

(4)在使用 SIMD NEON 汇编实现算法 4 的第 10−11 行计算泰勒公式或帕德近似的地方时，利用 mla 汇编指令和流水线原理，通过调整代码顺序，减少指令依赖，其原理与算法 2 相同。

## 4 性能评估

### 4.1 实验环境

#### 4.1.1 硬件环境搭建

本实验的测试环境基于华为鲲鹏 920 处理器，具体硬件测试环境如表 1 所列。

表 1 实验环境

Table 1 Experimental environment

| 操作系统 | CentOS Linux release 7.7.1908 |
|---|---|
| CPU 型号 | Kunpeng920 |
| 时钟频率/GHz | 2.6 |
| 内存容量/GB | 256 |
| L1 cache/KByte | 64 |
| L2 cache/KByte | 512 |
| CacheLine/Byte | 64 |

#### 4.1.2 软件环境搭建

本实验的对比对象是 libm 数学库和 ARM_M 算法库，实验在 ARM V8 架构上实现。通过将这两个算法库中的三角函数和利用 SIMD NEON 高性能实现并优化后的函数作对比来判断性能大小。

### 4.2 性能分析

我们的三角函数和 libm 数学库，Arm Performance Library 和 ARM_M 算法库中函数的计算时间对比结果如表 2 所列。

表 2 性能对比

Table 2 Performance comparison

| 函数 | 计算 65536 个元素的总时间/s | 本文工作/ms | libm/ms | ARM_M/ms |
|---|---|---|---|---|
| 单精度 sin | 0.182430 | 2784 | 10169 | 4115 |
| 双精度 sin | 0.512800 | 7825 | 27475 | 10488 |
| 单精度 cos | 0.195780 | 2987 | 10067 | 4502 |
| 双精度 cos | 0.563590 | 8600 | 33290 | 11676 |
| 单精度 tan | 0.313390 | 4782 | 28728 | ***N/A*** |
| 双精度 tan | 1.065850 | 16264 | 55021 | ***N/A*** |
| 单精度 atan | 0.209600 | 3198 | 20024 | ***N/A*** |
| 双精度 atan | 1.304160 | 19900 | 38004 | ***N/A*** |
| 单精度 atan2 | 0.422050 | 6440 | 35861 | ***N/A*** |
| 双精度 atan2 | 1.925440 | 29380 | 52167 | ***N/A*** |

表 2 第 1 列数据是本文工作中每个函数计算 65536 个元素的总时间(单位为 s)，后 3 列数据是本文工作、libm 和 ARM_M 中每个函数计算 1 个元素的时间(ns/elem)(单位为 ms)。

由表 2 可以看到，本文工作相对于 libm 算法库在时间性能上提升了 1.77～6.26 倍，可以注意到单精度函数优化效果比双精度函数更为明显。而本文工作相对于 ARM_M 算法库在时间性能上提升了 1.34～1.5 倍。

我们的工作在基于 ARM 架构的鲲鹏 920 处理器上测试的计算精度如表 3 所列。

表 3 计算精度

Table 3 Calculation accuracy

| 函数 | 平均误差 | 最大误差 |
|---|---|---|
| 单精度 sin | $2.24\times10^{-9}$ | $1.22\times10^{-7}$ |
| 双精度 sin | 0 | $2.20\times10^{-16}$ |
| 单精度 cos | $2.31\times10^{-8}$ | $1.41\times10^{-7}$ |
| 双精度 cos | 0 | $2.20\times10^{-16}$ |
| 单精度 tan | $1.22\times10^{-8}$ | $2.04\times10^{-7}$ |
| 双精度 tan | $2.00\times10^{-17}$ | $3.30\times10^{-16}$ |
| 单精度 atan | $5.63\times10^{-8}$ | $7.54\times10^{-7}$ |
| 双精度 atan | $2.00\times10^{-17}$ | $4.40\times10^{-16}$ |
| 单精度 atan2 | $3.83\times10^{-8}$ | $7.59\times10^{-7}$ |
| 双精度 atan2 | $2.00\times10^{-17}$ | $4.40\times10^{-16}$ |

由表 3 可以看出，单精度函数的平均误差均小于 $6.00\times10^{-8}$，而且最大误差不超过 $8.00\times10^{-7}$。双精度函数的平均误差均低于 $2.00\times10^{-17}$，且 sin，cos 的双精度函数的平均误差均为 0，而且最大误差不超过 $4.40\times10^{-16}$。

**结束语** 本文总结了基于 SIMD 的三角函数高性能实现和优化原理与性能，并将我们的工作与已有的 libm 数学库和 ARM_M 算法库进行比较，其总体性能具有优势。本文针对三角函数的实现主要是运用 SIMD NEON 汇编语言，在 ARM 架构上进行并行化。之所以使用这项技术一方面是由于目前 SIMD 技术具有重要地位，另一方面是由于 ARM 架构在市场上的份额逐步提高，其有着构建完整软件生态的迫切需要。因此基于 ARM 架构的三角函数的开发很有价值。当然不可避免地可以看到，有的三角函数的性能还有很大的优化空间，这也是我们下一步的工作。今后的三角函数的开发并不会局限于文中的 5 个三角函数，我们会继续研究开发其他三角函数，如反三角函数和算数函数等。

## 参 考 文 献

[1] FU S Y, WU J J, HSU W C. Improving SIMD code generation in QEMU[C]// 2015 Design, Automation & Test in Europe Conference & Exhibition(DATE). IEEE, 2015: 1233-1236.

[2] SHIBATA N. Efficient evaluation methods of elementary functions suitable for SIMD computation[J]. Computer Science-Research and Development, 2010, 25(1): 25-32.

[3] STEPHENS N, BILES S, BOETTCHER M, et al. The ARM scalable vector extension[J]. IEEE Micro, 2017, 37(2): 26-39.

[4] CHEN S M, GUO S Z, CHEN J X, et al. Optimization Algorithm for Trigonometric Functions Based on Processor with SIMD Function Components[J]. Journal of Information Engineering University, 2011, 12(1): 103-106.

[5] CAO D,GUO S Z,ZHANG X. Implementation and Optimization of Extended Function Library Based on SW26010 Processor[J]. Computer Engineering,2017(1):61-66.

[6] LI Q Y,WANG N C,YI D Y. Numerical analysis 5th edition [M]. Tsinghua University Press,2008.

[7] HACKBUSCH W. Computation of best $ $ L^{\\infty} $ $ L∞ exponential sums for $1/x$ by Remez' algorithm[J]. Computing and Visualization in Science,2019,20(1):1-11.

[8] GLUZMAN S,YUKALOV V I. Self-similarly corrected Pade approximants for nonlinear equations[J]. International Journal of Modern Physics B,2019,33(29):1950353.

[9] HE Z,ZHANG J,YAO Z. Determining the optimal coefficients of the explicit finite difference scheme using the Remez exchange algorithm[J]. Geophysics,2019,84(3):S137-S147.

[10] MACCHIARELLA G. "Equi-Ripple" Synthesis of Multiband Prototype Filters Using a Remez-Like Algorithm[J]. IEEE Microwave and Wireless Components Letters,2013,23(5):231-233.

[11] VINSCHEN C,JOHNSTON J. Standard C math library[OL]. https://www.sourceware.org/newlib/libm.html.

[12] ARM. Arm Performance Libraries Reference Guide [OL]. https://static.docs.arm.com/101004/1920/arm_performance_libraries_reference_101004_1920_00_en.pdf.

[13] ARM. Arm Optimized Routines [OL]. https://github.com/ARM-software/optimized-routines.

[14] ZHA Y L. Qin Jiushao's mathematical thinking method[J]. Research on Dialectics of Nature,2003,19(1):87-92.

**YAO Jian-yu**, born in 1997, postgraduate. His main research interests include high performance computing and parallel software, etc.

**ZHANG Guang-ting**, born in 1987, MS, assistant professor, is a member of China Computer Federation. Her main research interests include high performance computing and parallel software.