圆周率计算背景下的 C++ 与 Haskell 语言比较

张罗亮

2025 年 11 月 20 日

摘要

高效计算圆周率（π）是检验算法与编程语言的经典问题。然而，对于初学者而言，理解不同编程范式下的实现方式及其效率差异存在挑战。本文旨在解决初学者在选择编程语言时缺乏直观参考的问题，通过使用 C++（指令式范式）和 Haskell（函数式范式），分别实现基于 Leibniz 级数和 Machin 公式的圆周率计算方法，从代码风格与运行效率两个维度进行系统对比分析。实验结果表明，Machin 公式在收敛速度上显著优于 Leibniz 级数，而 C++ 在运行效率上普遍高于 Haskell，但在代码表达简洁性方面 Haskell 更具优势。本文为初学者理解不同编程范式的特点及进行语言选择提供了直观的参考依据。

1 引言

在程序设计语言的学习中，理解不同编程范式（如指令式与函数式）的差异是一个重要的环节。圆周率（π）的计算问题因其算法经典、实现直观，成为进行语言对比的理想案例。

对于初学者而言，通过完成相同的计算任务来感受不同语言的特点，是一种有效的学习方式。然而，现有的学习资料往往缺乏将不同范式的实现代码及其性能进行并置对比的直观案例。

为此，本文选择 C++ 和 Haskell 作为指令式与函数式范式的代表，以实现 Leibniz 级数和 Machin 公式计算圆周率为具体任务。本文的核心目的在于：

- 展示实现差异：通过并排展示两种语言实现相同算法的代码，直观呈现其在语法和结构上的不同。

- 比较运行效率：在相同计算任务下，测量并对比两种语言实现的运行时间，获得初步的性能认知。
- 提供直观参考：基于代码风格和运行时间的简单对比，为初学者在选择语言解决类似数值计算问题时提供参考。

下文将首先介绍两种算法的数学原理与实现代码，随后展示并分析其运行效率的测试结果，最后给出总结。

2 用编程语言计算 π

2.1 Leibniz 级数

2.1.1 Leibniz 级数

$$\pi = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}.$$

2.1.2 C++

```cpp
#include <iostream>
#include <iomanip>
#include <cmath>
double calculatePiLeibniz(int iterations) {
    double pi = 0.0;
    for (int i = 0; i < iterations; i++) {
        double term = (i % 2 == 0) ? 1.0 : -1.0;
        pi += term / (2 * i + 1);
    }
    return pi * 4;
}
int main() {
    int iterations;
    std::cin >> iterations;
    double pi = calculatePiLeibniz(iterations);
    std::cout << std::fixed << std::setprecision(20) << pi << std::endl;
    return 0;
}
```

Listing 1: Leibniz 级数计算

2.1.3 Haskell

```haskell
module Main where
import Text.Printf
calculatePiLeibniz :: Int -> Double
calculatePiLeibniz iterations = 4 * sum [term i | i <- [0..iterations-1]]
    where
    term i = if even i then 1 / fromIntegral (2*i + 1) else -1 /
    fromIntegral (2*i + 1)
main :: IO ()
main = do
    let iterations = 1000000
    let pi = calculatePiLeibniz iterations
    printf "%.20f\n" pi
```

Listing 2: Leibniz 级数计算

2.1.4 两种编程语言的对比

对求和符号 $\sum$ 的处理： C++ 采用直截了当的循环，符合直觉；Haskell 使用对闭区间 $[0, \text{iterations} - 1]$ 的枚举，较为隐晦。

对单个项的计算： 对于 $(-1)^n$，要判断 n 的奇偶性，C++ 可以使用条件运算符，Haskell 可以使用 `if then else` 语句，形式不同，实质相同。

表 1: Leibniz 级数计算 π 值的精度和性能比较

语言	迭代次数	计算值	绝对误差	时间 (秒)
C++	100000	3.14158265358971977577	1.00e-5	0.015132
Haskell	100000	3.14158265358971980000	1.00e-5	0.042757
C++	1000000	3.14159165358977432447	1.00e-6	0.016325
Haskell	1000000	3.14159165358977430000	1.00e-6	0.181700
C++	10000000	3.14159255358979150330	1.00e-7	0.037844
Haskell	10000000	3.14159255358979150000	1.00e-7	1.605598
C++	100000000	3.14159264358932599492	1.00e-8	0.256115
Haskell	100000000	3.14159264358932600000	1.00e-8	16.436355

2.2 Machin 公式

2.2.1 Machin 公式

$$\frac{\pi}{4} = 4\arctan\frac{1}{5} - \arctan\frac{1}{239}$$

2.2.2 C++

```cpp
#include <boost/multiprecision/cpp_dec_float.hpp>
#include <iostream>
#include <iomanip>
using namespace boost::multiprecision;
using mp_type = number<cpp_dec_float<1000>>;
mp_type arctan_taylor(const mp_type& x, int terms) {
    mp_type result = 0;
    mp_type x_power = x;
    mp_type x_squared = x * x;
    for (int n = 0; n < terms; ++n) {
        mp_type term = x_power / (2 * n + 1);
        if (n % 2 == 0) result += term;
        else result -= term;
        x_power *= x_squared;
    }
    return result;
}
mp_type calculate_pi_machin(int terms) {
    mp_type one = 1;
    mp_type arctan_1_5 = arctan_taylor(one / 5, terms);
    mp_type arctan_1_239 = arctan_taylor(one / 239, terms);
    return 4 * (4 * arctan_1_5 - arctan_1_239);
}
int main() {
    int terms;
    std::cin >> terms;
    mp_type pi = calculate_pi_machin(terms);
    std::cout << std::fixed << std::setprecision(1000) << pi << "\n";
    return 0;
}
```

Listing 3: Machin 公式计算

2.2.3 Haskell

```haskell
{-# LANGUAGE OverloadedStrings #-}
import Data.Number.CReal
import Text.Printf
arctanTaylorHP :: CReal -> Int -> CReal
arctanTaylorHP x iterations = sum [term n | n <- [0..iterations-1]] where
    term n = let coefficient = if even n then 1 else -1
             in coefficient * (x^(2*n+1)) / fromIntegral (2*n+1)
calculatePiMachinHP :: Int -> CReal
calculatePiMachinHP iterations = 4 * (4 * arctan_1_5 - arctan_1_239) where
    arctan_1_5 = arctanTaylorHP (1/5) iterations
    arctan_1_239 = arctanTaylorHP (1/239) iterations
main :: IO ()
main = do
    input <- getLine
    let iterations = read input
    let piHP = calculatePiMachinHP iterations
    putStrLn $ showCReal 1000 piHP
```

Listing 4: Machin 公式计算

2.2.4 两种编程语言的对比

表 2: Machin 公式计算 π 值的精度和性能比较

语言	迭代次数	计算值	绝对误差	时间 (秒)
C++	200	3.14159265358979323846	1.98e-282	0.038422
Haskell	200	3.14159265358979323846	1.98e-282	3.082227
C++	500	3.14159265358979323846	3.29e-702	0.077586
Haskell	500	3.14159265358979323846	3.29e-702	22.373403
C++	800	3.14159265358979323846	0.00e-998*	0.115542
Haskell	800	3.14159265358979323846	0.00e-998*	64.167978

*0.00e-998 表示在计算精度范围内，未发现误差。

3 数学方法的简单证明

3.1 Leibniz 级数

3.1.1 Leibniz 级数

$$\pi = 4\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}.$$

3.1.2 Leibniz 级数的证明

我们知道：

$$\frac{\mathrm{d}}{\mathrm{d}x}\arctan(x) = \frac{1}{1+x^2}.$$

对于 $|t|<1$，有：

$$\frac{1}{1-t} = \sum_{n=0}^{\infty} t^n.$$

令 $t=-x^2$，则当 $|x|<1$ 时：

$$\frac{1}{1+x^2} = \sum_{n=0}^{\infty}(-1)^n x^{2n}.$$

积分得到：

$$\arctan(x) = \int_0^x \frac{1}{1+t^2}\,\mathrm{d}t = \int_0^x \sum_{n=0}^{\infty}(-1)^n t^{2n}\,\mathrm{d}t.$$

在 $|x|<1$ 时可逐项积分：

$$\arctan(x) = \sum_{n=0}^{\infty}(-1)^n\frac{x^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

当 $x=1$ 时：

$$\frac{\pi}{4} = \arctan(1) = \sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1}.$$

3.1.3 Leibniz 级数的收敛速度

对于交错级数，若 $S_N = \sum\limits_{n=0}^{N} a_n$ 是部分和，且 a_n 满足 $|a_n|$ 单调递减趋于 0，则有误差估计：

$$|S - S_N| \le |a_{N+1}|.$$

在这里 $a_n = \frac{(-1)^n}{2n+1}$，所以：

$$\left| \frac{\pi}{4} - \sum_{n=0}^{N} \frac{(-1)^n}{2n+1} \right| \leq \frac{1}{2N+3}.$$

要保证误差不超过 $\epsilon > 0$，即：

$$\frac{1}{2N+3} \leq \epsilon.$$

解此不等式：

$$N \geq \frac{\frac{1}{\epsilon} - 3}{2} = \frac{1}{2\epsilon} - \frac{3}{2}.$$

因此，所需的最小项数为：

$$N_{\min} = \left\lceil \frac{1}{2\epsilon} - \frac{3}{2} \right\rceil.$$

Leibniz 级数收敛，但仅为线性收敛（误差量级为 $\frac{1}{N}$），计算 π 的效率很低，更多用于理论分析和数学教学。

3.2 Machin 公式

3.2.1 Machin 公式

$$\frac{\pi}{4} = 4 \arctan \frac{1}{5} - \arctan \frac{1}{239}$$

3.2.2 Machin 公式的证明

令 $\alpha = \arctan \frac{1}{5}$，$\beta = \arctan \frac{1}{239}$。

1. $\tan 2\alpha = \dfrac{2 \cdot \frac{1}{5}}{1 - (\frac{1}{5})^2} = \dfrac{5}{12}$
2. $\tan 4\alpha = \dfrac{2 \cdot \frac{5}{12}}{1 - (\frac{5}{12})^2} = \dfrac{120}{119}$
3. $\tan(4\alpha - \beta) = \dfrac{\frac{120}{119} - \frac{1}{239}}{1 + \frac{120}{119} \cdot \frac{1}{239}} = \dfrac{\frac{120 \cdot 239 - 119}{119 \cdot 239}}{\frac{119 \cdot 239 + 120}{119 \cdot 239}} = \dfrac{\frac{28561}{119 \cdot 239}}{\frac{28561}{119 \cdot 239}} = 1$

$$\tan(4\alpha - \beta) = 1 \quad \Rightarrow \quad 4\alpha - \beta = \frac{\pi}{4}$$

3.2.3 Machin 公式的收敛速度

用泰勒展开：

$$\arctan x = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} x^{2k+1}$$

1. 第一项 $4\arctan(\frac{1}{5})$ 误差主项为 $\frac{4}{(2n+1)5^{2n+1}}$
2. 第二项 $\arctan(\frac{1}{239})$ 误差主项为 $\frac{4}{(2n+1)239^{2n+1}}$，收敛极快，可忽略其误差

要达到 d 位十进制精度，需：

$$\frac{4}{(2n+1)5^{2n+1}} < 10^{-d} \quad \Leftrightarrow \quad 5^{2n+1} > \frac{4 \cdot 10^d}{2n+1} \Leftarrow 5^{2n} > 10^d \Leftarrow n > \frac{\log_5(10)}{2} \cdot d > 0.7d$$

Machin 公式项数与精度位数成线性关系，比 Leibniz 级数快指数级。

4 结论

本文通过实现 Leibniz 级数和 Machin 公式，对比分析了 C++ 和 Haskell 在计算圆周率任务中的表现，得出以下结论：

- **在算法层面，收敛速度是关键**。对于计算圆周率（π），在精度要求不变时，想要加快计算，首要的方式是选取收敛更快的关于 π 的级数。
- **在实现层面，编程语言范式导致巨大性能差异**。在相同的算法和精度下，C++ 明显快于 Haskell，这可能是因为：
  - C++ 的编译优化可以生成非常高效的机器码。
  - Haskell 的纯函数式、惰性求值以及不可变数据结构等特性，导致了额外的内存分配和间接开销，这在密集的数值循环中会被放大。
- **在代码风格上，两种语言反映了不同的设计哲学**。C++ 代码直述“如何做”，步骤清晰，符合传统编程直觉。Haskell 代码则声明“是什么”，更贴近数学公式的表述，简洁而高雅。