**北京邮电大学软件学院**

**2019-2020学年第一学期实验报告**

**课程名称：** 并行计算

**项目名称：** 圆周率π的计算

**项目完成人：**

**姓名：** 徐龙 **学号：**  2018522108

**指导教师：** 卢本捷

**日 期： 2020年 11 月 1 日**

1. **实验目的**

学习并行计算的初步方法。

1. **实验内容**

用多种方法完成pi的并行计算

1. **实验环境**
2. 两台或以上的windows 或linux等。
3. 节点的网络互联。
4. Visual studio 2017
5. **实验要求**
6. 记录实验过程，分析试验现象。
7. 记录主要源代码
8. 撰写实验报告.
9. **MPICH 实验步骤**

本次实验一共实现了7种计算π的方法，其中每种方法被封装到一个函数中，每测试一个函数，就在main函数中调用该函数，main函数中调用MPI\_Bacst()和MPI\_Reduce()负责分发和汇总每个进程计算PI的任务,然后将结果打印出来。

1. 每个函数使用的全局变量如下：

//记录每个进程所计算结果的和

long double ValueSum = 0.0;

//进程数量,进程ID

int Size, MyID;

//PI计算累加次数, 1000000

long double N = 1e7;

1. 每种计算PI的方法被封装的函数名如下：

    //1.面积积分

    areaIntegral();

    //2.幂级数积分

    powerSeries();

    //3.改进幂级数积分

    powerSeriesImprove();

    //4.蒙特卡洛的方式

    MonteCarlo();

    //5.金田康正的方法

    JinTianKangZheng();

    //6.Chudnovsky方法 世界纪录

    Chudnovsky();

1. **面积积分**
2. 问题描述：

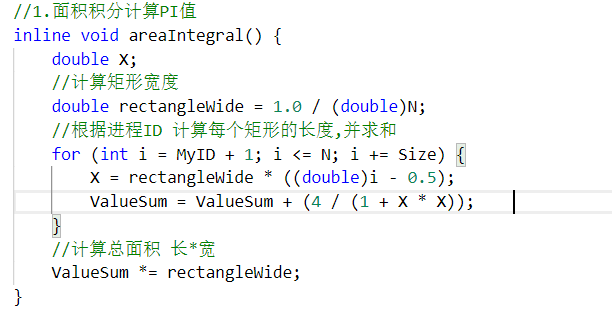
对积分进行数值求解即可。

对区间进行划分。简单并行计算。

这是MPICH2中提供的示例方式。

1. 解法：

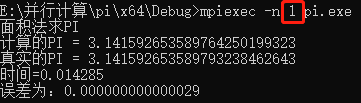
* 采用等宽矩形面积求和的方法代替积分。
* 将积分面积分为N个小块。
* 由主进程将计算N个小块面积的任务平均分给其他Size个进程。
* 汇集Size个进程的计算结果，将结果累加在一起。
* 输出结果。

1. **代码实现**

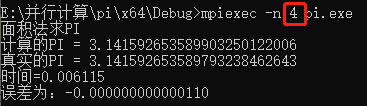


1. **运行结果**

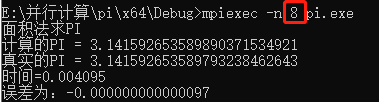
当n = 1000000，进程数为1时



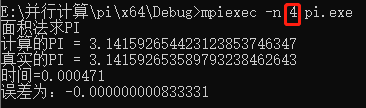
当n = 1000000，进程数为4时



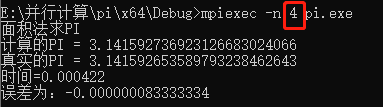
当n = 1000000，进程数为8时



当n = 10000，进程数为4时



当n = 1000，进程数为4时



## 结果比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试用例 | n | 进程数 | 运行时间 | 误差 |
| 1 | 1000000 | 1 | 0.014285 | 0.000000000000029 |
| 2 | 1000000 | 4 | 0.006115 | -0.000000000000110 |
| 3 | 1000000 | 8 | 0.004095 | -0.000000000000097 |
| 4 | 10000 | 4 | 0.000471 | -0.000000000833331 |
| 5 | 1000 | 4 | 0.000422 | -0.000000083333334 |

通过比较发现：

1. 当n不变，进程数增多时，运行时间会减小。
2. 当进程数不变，积分区域划分数越多时，误差会减小。

# 方法二：幂级数

## 问题描述

因为tan（ /4）=1，故 /4= arctan（1）,对arctan（1）进行幂级数展开即可：

arctgx 计算积分得：argtgx=

所以当x=1 时：

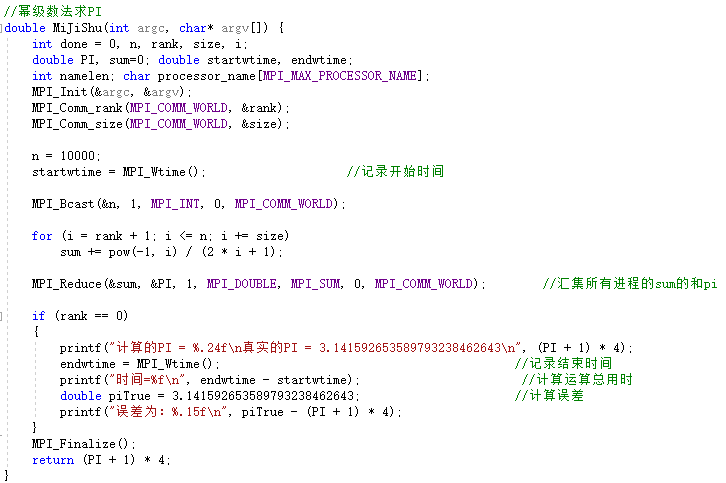
/4= = 1-1/3+1/5-1/7+……+ (-1)n/2n+1

## 解法

并行计算方法：

1. 事先确定进程数。
2. 主进程确定计算的项数。向各个子进程发送项数。
3. 各个子进程进行自己这部分的累加。
4. 主进程集中。
5. 记录运算时间。

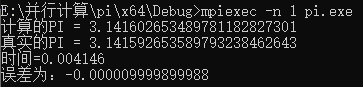
## 代码实现



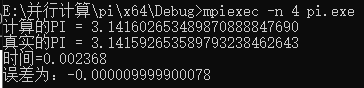
代码实现过程为先初始化进程，n为多项式的个数，使用MPI\_Bcast将n广播给所有进程，然后按照公式计算进程负责的各自的多项式，然后采用MPI\_Reduce()函数将所有进程的结果汇总相加，最后输出求出的PI，并计算运算时间和误差。

## 运行结果

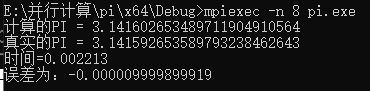
当n = 100000，进程数为1时



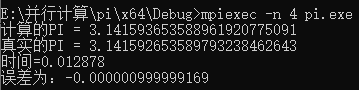
当n = 100000，进程数为4时



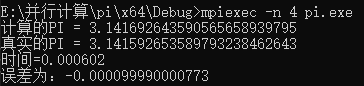
当n = 100000，进程数为8时



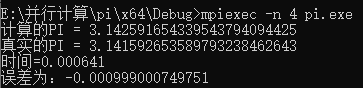
当n = 1000000，进程数为4时



当n = 10000，进程数为4时



当n = 1000，进程数为4时



## 结果比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试用例 | n | 进程数 | 运行时间 | 误差 |
| 1 | 100000 | 1 | 0.004146 | -0.000009999899988 |
| 2 | 100000 | 4 | 0.002368 | -0.000009999900078 |
| 3 | 100000 | 8 | 0.002213 | -0.000009999899919 |
| 4 | 1000000 | 4 | 0.012878 | -0.000000999999169 |
| 5 | 10000 | 4 | 0.000602 | -0.000099990000773 |
| 6 | 1000 | 4 | 0.000641 | -0.000999000749751 |

通过比较发现：

1. 当n不变，进程数增多时，运行时间会减小。
2. 当进程数不变，多项式数越多时，误差会减小，运行时间也会增大。

# 方法三：改进的幂级数

## 问题描述

方法一和方法二收敛很慢。要精确到 大致需要计算2× 项。为提高计算速度采用以下改进的方法：

对于幂级数而言，当x越接近于0时，收敛越快。 上面的例子中，x=1，离0有相当的距离。

令x=1/5. 记=arctan(1/5). tan =1/5.

tan2= 2tan/(1-tan2) = 5/12.

同理tan4 = 120/119. 而tan（ /4）=1，可见 4 与 /4 非常接近。

令 =4- /4

所以tan =tan(4- /4) = （根据 tan(A-B) = (tanA-tanB)/(1+tanAtanB)）

=arctg (1/239).

所以： /4 =4 – = 4 × arctg(1/5)- arctg(1/239)

再利用幂级数展开：=4 -

上述级数收敛的速度非常快。

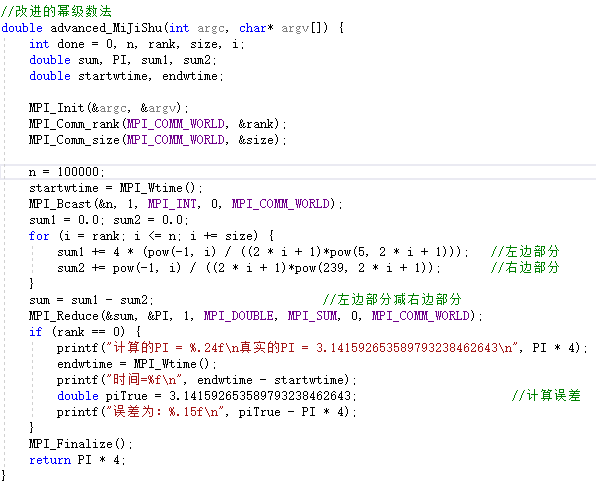
左边部分：当n=4时，即有1/9×59 < 10-6，而右边收敛更快。

## 解法

并行计算方法：

1. 事先确定进程数。
2. 主进程确定计算的项数。向各个子进程发送项数。
3. 各个子进程进行自己这部分的累加。
4. 主进程集中。
5. 记录运算时间。

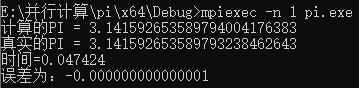
## 代码实现



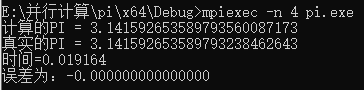
代码实现过程为先初始化进程，n为一组相减的多项式的个数，使用MPI\_Bcast将n广播给所有进程，然后按照公式计算分别计算多项式的左部和右部，然后左部减去右部，采用MPI\_Reduce()函数将所有进程的结果汇总相加，最后输出求出的PI，并计算运算时间和误差。

## 运行结果

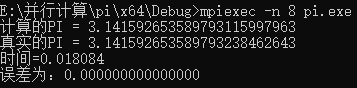
当n = 100000，进程数为1时



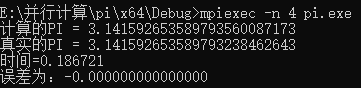
当n = 100000，进程数为4时



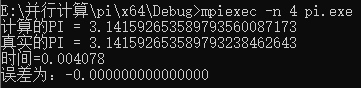
当n = 100000，进程数为8时



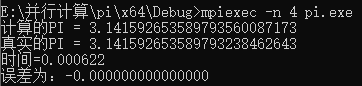
当n = 1000000，进程数为4时



当n = 10000，进程数为4时



当n = 1000，进程数为4时



## 结果比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试用例 | n | 进程数 | 运行时间 | 误差 |
| 1 | 100000 | 1 | 0.047424 | -0.000000000000001 |
| 2 | 100000 | 4 | 0.019164 | -0.000000000000000 |
| 3 | 100000 | 8 | 0.018084 | 0.000000000000000 |
| 4 | 1000000 | 4 | 0.186721 | -0.000000000000000 |
| 5 | 10000 | 4 | 0.004078 | -0.000000000000000 |
| 6 | 1000 | 4 | 0.000622 | -0.000000000000000 |

通过比较发现：

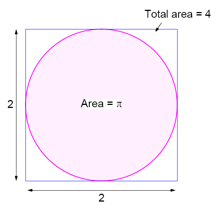
1. 这种方法的精确度确实比方法一和方法二高。
2. 当n不变，进程数增多时，运行时间会减小，但是我发现因为我的cpu有4核，当进程数大于4的时候，运行时间的减少不再明显。
3. 当进程数不变，多项式越多时，误差会减小，但是运行时间会增大。

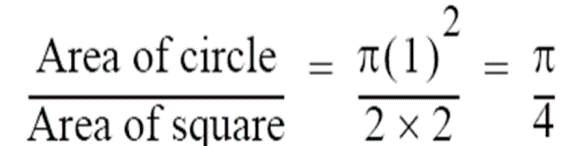
# 方法四：蒙特卡洛方式

## 问题描述

1. 各个子进程的工作
2. 使用随机数在正方形内投点
3. 计算落在圆内的点的次数。
4. 计算比值。
5. 主进程收集所有的结果，进行平均。

## 解法

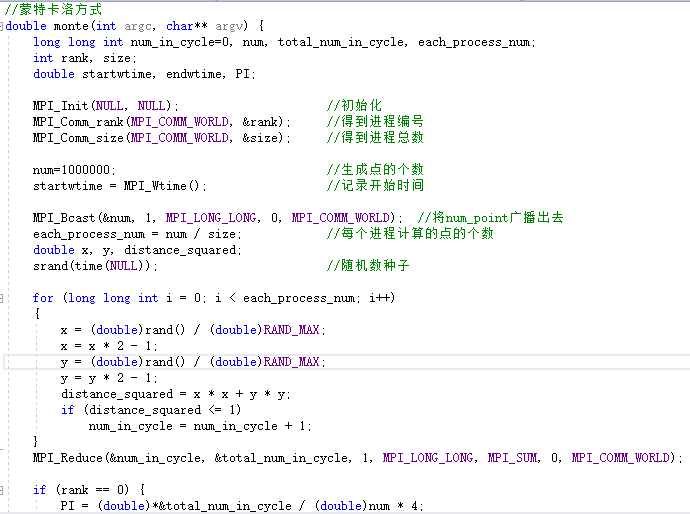


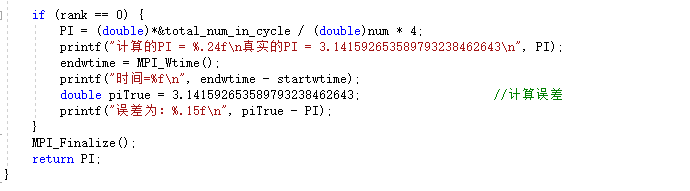


用概率的方式计算pi

1. 在2×2矩形范围内随机产生一个二维坐标，检测其是否落入圆内。
2. 累计落入圆内的总数，和所有点的总数。
3. 其比值即pi/4
4. 每个处理器各自计算，求均值即可。

## 代码实现

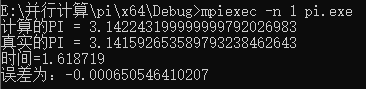




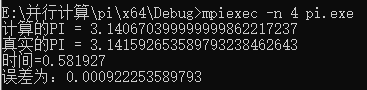
代码实现过程为先初始化进程，n是总共要产生点的个数，使用MPI\_Bcast()将n广播给所有进程，然后给每个进程分配点数，生成随机数，计算生成的点是否在圆内，此圆以坐标原点为圆心，点到圆心距离小于1则在圆内，记录在圆内点的个数，采用MPI\_Reduce()函数将所有进程的结果汇总相加，然后除以总的点的个数，再乘以4，即为PI的值，最后输出求出的PI，并计算运算时间和误差。

## 运行结果

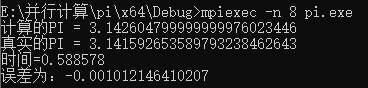
当n = 10000000，进程数为1时



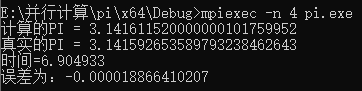
当n = 10000000，进程数为4时



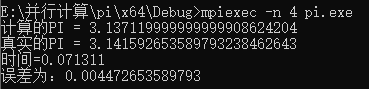
当n = 10000000，进程数为8时



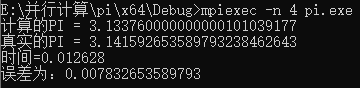
当n = 100000000，进程数为4时



当n = 1000000，进程数为4时



当n = 100000，进程数为4时



## 结果比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试用例 | n | 进程数 | 运行时间 | 误差 |
| 1 | 10000000 | 1 | 1.618719 | -0.000650546410207 |
| 2 | 10000000 | 4 | 0.581927 | 0.000922253589793 |
| 3 | 10000000 | 8 | 0.588578 | -0.001012146410207 |
| 4 | 100000000 | 4 | 6.904933 | -0.000018866410207 |
| 5 | 1000000 | 4 | 0.071311 | 0.004472653589793 |
| 6 | 100000 | 4 | 0.012628 | 0.007832653589793 |

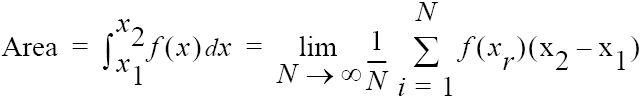
通过比较发现：

1. 这种方法的精确度不算很高。
2. 当n不变，进程数增多时，运行时间会减小。
3. 当进程数不变，产生的点越多，误差会减小，但是运行时间会增大。

# 方法五：随机积分法

## 问题描述

利用公式：

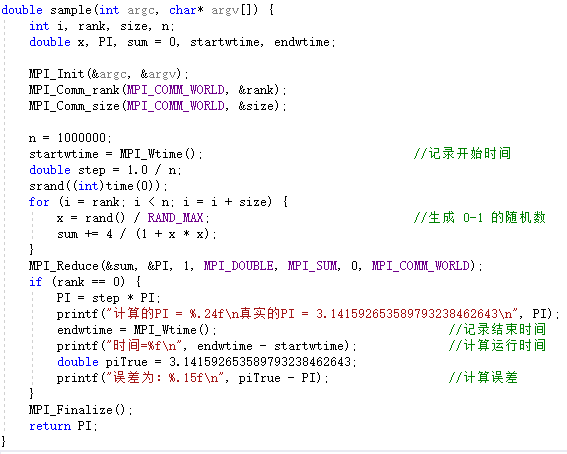


计算

## 解法

将式子转化为上述的形式，计算在0~1之间的随机样本的函数值，最后相加。

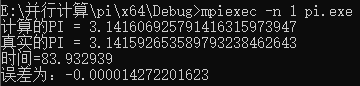
## 代码实现



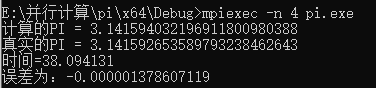
代码实现过程为先初始化进程，n是总共要产生随机点的个数，使用MPI\_Bcast()将n广播给所有进程，然后给每个进程分配点数，生成0~1的随机数，将随机数的函数值相加，采用MPI\_Reduce()函数将所有进程的结果汇总相加，，最后输出PI，并计算运算时间和误差。

## 运行结果

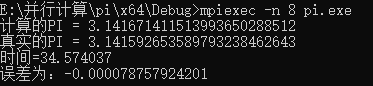
当n = 1000000000，进程数为1时



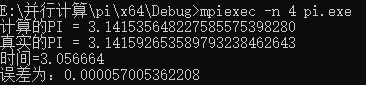
当n = 1000000000，进程数为4时



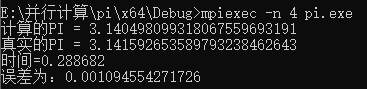
当n = 1000000000，进程数为8时



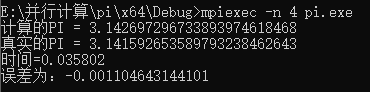
当n = 100000000，进程数为4时



当n = 10000000，进程数为4时



当n = 1000000，进程数为4时



## 结果比较

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 测试用例 | n | 进程数 | 运行时间 | 误差 |
| 1 | 1000000000 | 1 | 83.932939 | -0.000014272201623 |
| 2 | 1000000000 | 4 | 38.094131 | -0.000001378607119 |
| 3 | 1000000000 | 8 | 34.574037 | -0.000078757924201 |
| 4 | 100000000 | 4 | 3.056664 | 0.000057005362208 |
| 5 | 10000000 | 4 | 0.288682 | 0.001094554271726 |
| 6 | 1000000 | 4 | 0.035802 | -0.001104643144101 |

通过比较发现：

1. 相比其他方法，这种方法的精确度不算很高，而且运行时间较长。
2. 当n不变，进程数增多时，运行时间会减小。
3. 当进程数不变，产生的随机点越多，误差会减小，但是运行时间会大幅度增大。
4. **调试心得**

通过本次实验，我初步掌握了并行计算的基本思想，并且学会了如何使用mpi库的基础函数，使用多种方法求解pi的值，并且最终将程序运行成功，并对实验结果进行了一系列的分析。