《并行计算》实验报告（正文）

姓名____学名_____完成时间____25/3/11_____

一、实验名称与内容

多线程计算正弦值：

利用正弦函数的泰勒级数展开式计算结果：

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

要求：Pthread 并行化实现。

输入包含线程数和弧度值。

二、实验环境的配置参数

（CPU/GPU 型号与参数、内存容量与带宽、互联网络参数等）

操作系统：国家超级计算天津中心定制操作系统

CPU：国产飞腾处理器

单核理论性能（双精度）9.2GFlops

单节点理论性能（双精度）588.8GFlops

内存：128G

网络：天河自主高速互联网络（400GB/s）

编译环境：GCC9.3.0：gcc，g++，gfort 等

三、实验题目问题分析

（本题目是什么类型的计算问题，可以有哪些并行化方案等）

（1）这是计算密集型数值计算问题，可以将数据划分并行。

（2）将泰勒级数的前 N 项均匀分配给多个线程，每个线程独立计算其分配项的值。

（3）本实验采取循环划分的方式，即每个线程按固定步长跳跃式处理不同的项。

例如：若线程数为 4（step=4）：

线程 0：处理 i=0, 4, 8, ...

线程 1：处理 i=1, 5, 9, ...

线程 2：处理 i=2, 6, 10, ...

线程 3：处理 i=3, 7, 11, ...

（4）每个线程先计算局部和 val，最后再通过互斥锁合并到 res

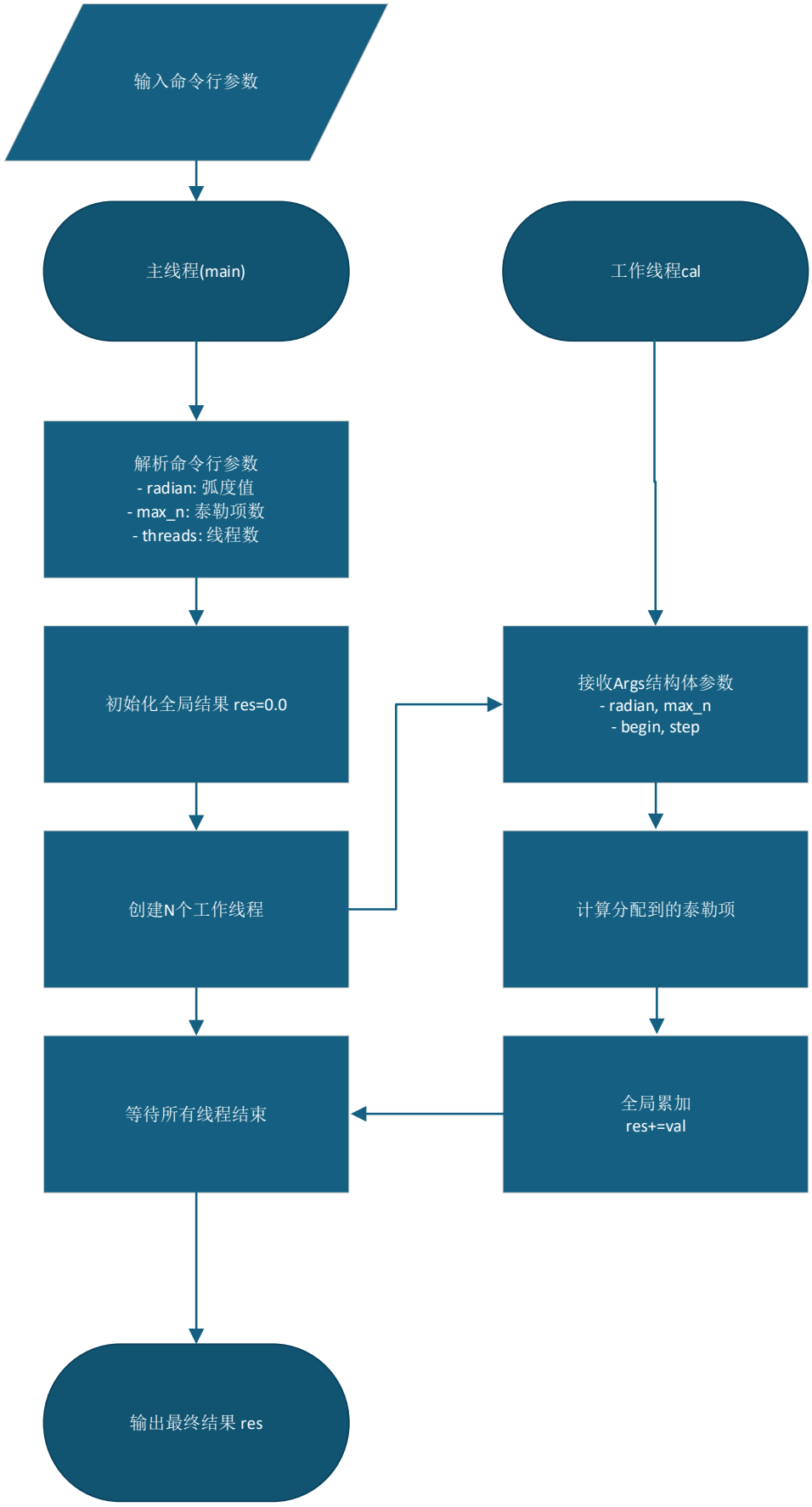
四、方案设计

（以流程图、框图等方式给出并行方法的思路，以伪代码方式给出并行算法细节）

（1）大致思路：

首先通过命令行输入参数，然后在主线程中，解析参数，然后创建子线程，每个子线程计算自己的部分，并将结果累加到全局共享变量，然后主线程输出结果。

（2）流程图如下：


输入命令行参数
主线程(main)
工作线程cal
解析命令行参数
- radian: 弧度值
- max_n: 泰勒项数
- threads: 线程数
初始化全局结果 res=0.0
接收Args结构体参数
- radian, max_n
- begin, step
创建N个工作线程
计算分配到的泰勒项
等待所有线程结束
全局累加
res+=val
输出最终结果 res

（3）伪代码：

```
全局变量:
    res = 0.0                       // 计算结果
    mutex = 互斥锁初始化             // 保护共享变量res

结构体 Args:
    float radian                    // 弧度
    long max_n                      // 泰勒展开的最大项数
    long begin                      // 线程起始计算项
    long step                       // 线程计算步长

函数 main():
    if 参数数量 != 4:
        输出错误信息
        退出程序

    解析参数:
        radian = 第一个参数
        max_n = 第二个参数
        threads = 第三个参数

    线程数组 pid[threads]

    for i = 0 to threads-1:  // 创建线程
        分配 Args 结构体 arg
        参数:
            arg.radian = radian
            arg.max_n = max_n
            arg.begin = i        // 各线程起始点不同
            arg.step = threads   // 步长为总线程数
        创建线程 pid[i]，执行 cal 函数，传入arg

    for i = 0 to threads-1
        wait pid[i] //等待线程结束

    输出res
```

```
工作线程 cal( _arg):
    局部变量:
        val = 0.0    // 本线程的累加值
        tmp          // 临时存储单泰勒项的值

    for i = arg.begin to arg.max_n，步长 arg.step:
        tmp= 1.0
        计算泰勒项:
            for j = 1 to 2*i+1:
                tmp *= (arg.radian / j)  // 隐式计算 x^(2i+1)/(2i+1)!
        根据项的符号累加到val:
            if i是奇数: val -= tmp
            else:       val += tmp

    得到互斥锁:
        res += val
    解锁

    返回空指针
```

五、实现方法

（结合所用计算环境，给出具体的编程实现方案）

（1）下面是具体代码（sin1.c）：

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

double res;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

typedef struct Args {
    float radian;
    long max_n;
    long begin;
    long step;
} Args;


void *cal(void *arg);
```

```c
int main(int argc, char *argv[]) {
    if (argc != 4) {
        printf("Parameters error: radian N threads!\n");
        exit(1);
    }

    long i;
    double radian = atof(argv[1]);
    long max_n = atol(argv[2]);
    int threads = atoi(argv[3]);
    Args *arg;
    double* val;
    pthread_t *pid;
    pid = (pthread_t *)malloc(threads * sizeof(pthread_t));
    res = 0.0;
    for (i = 0; i < threads; i++) {
        arg = (Args *)malloc(sizeof(Args));
        arg->radian = radian;
        arg->max_n = max_n;
        arg->begin = i;
        arg->step = threads;
        pthread_create(&pid[i], NULL, cal, (void *)arg);
    }
    for (i = 0; i < threads; i++) {
        pthread_join(pid[i],NULL);
    }
    printf("%lf\n", res);
    free(pid);
    return 0;
}
```

```c
void *cal(void *_arg) {
    long i, j;
    double val, tmp;
    Args *arg = (Args *)_arg;

    val = 0.0;
    for (i = arg->begin; i <= arg->max_n; i += arg->step) {
        tmp = 1.0;
        for (j = 1; j <= 2*i+1; j++) {
            tmp *= arg->radian / j;
        }
        if (i % 2 > 0) {
            val -= tmp;
        } else {
            val += tmp;
        }
    }
    pthread_mutex_lock(&mutex);
    res += val;
    pthread_mutex_unlock(&mutex);
    return NULL;
}
```

（2）sin1.sh：为了得到测试结果，编写脚本文件

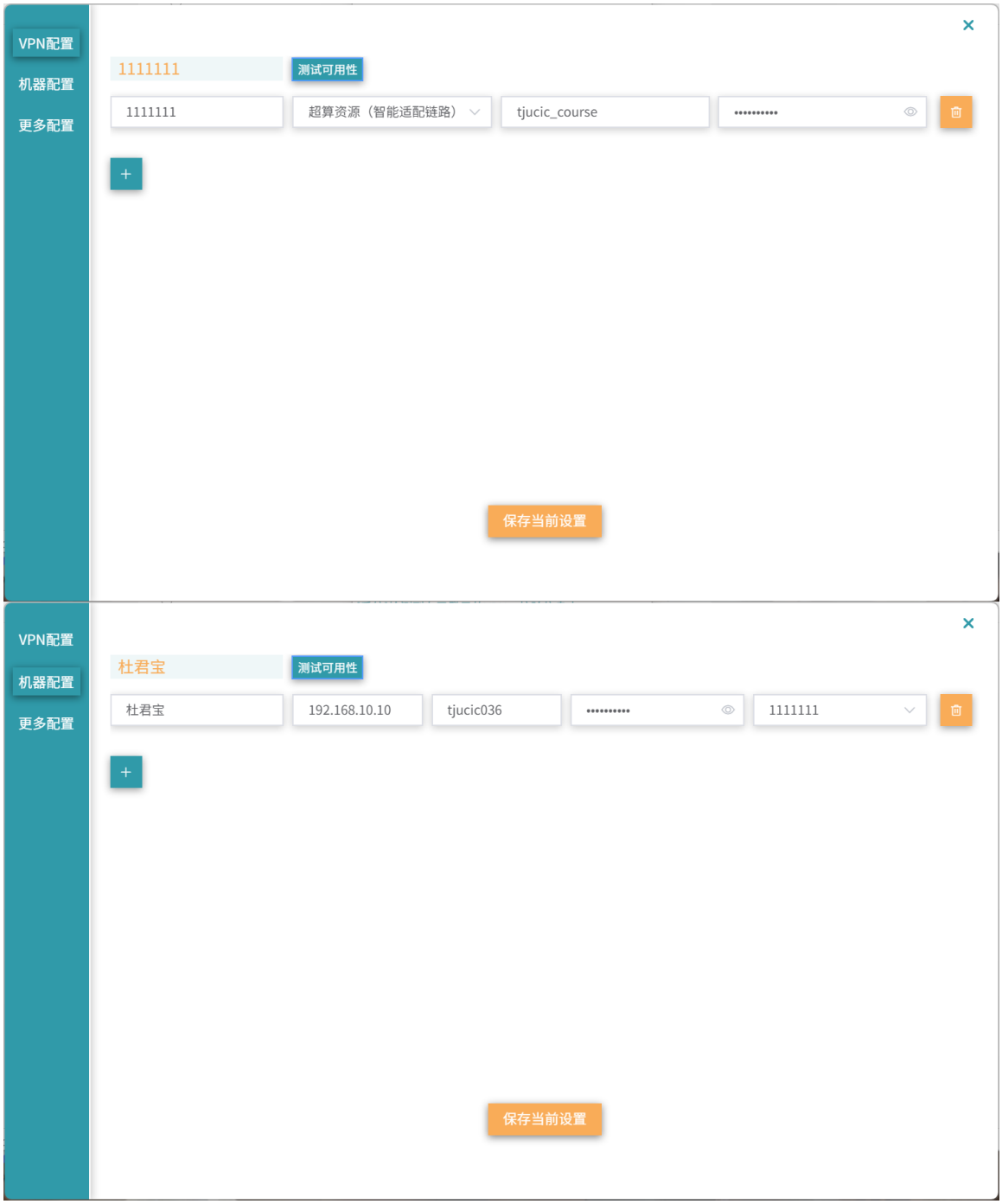
为了正确将监测的时间写入.log 文件，将输出重定向。同时，该脚本文件测试了多种线程下，求 x = 1，n = 100000 的运行时间。

```
#!/bin/bash

#清空上一次的作业日志和.o文件
rm slurm-*.out

# 清空旧日志
> sin1.log

run_test() {
    threads=$1
    echo "===== $threads thread(s) =====" >> sin1.log

    #time 的输出默认到 stderr，需要 2>&1 重定向
    { time yhrun -n 1 -c $threads sin1.o 1 100000 $threads 2>&1; } >> sin1.log 2>&1

    echo "" >> sin1.log  # 添加空行
}

#以不同线程数运行并测试时间
run_test 1
run_test 2
run_test 3
run_test 4
run_test 8
run_test 10
run_test 12
run_test 14
run_test 16
```

六、结果分析

（在结果正确的前提下，分析所实现方案的加速比、效率等指标）

首先下载青索客户端，分别将 vpn 和用户名及密码配置好：



然后将 sin1.c 文件和 sin1.sh 文件上传到超算平台，打开命令行，输入指令编译，并上传作业：

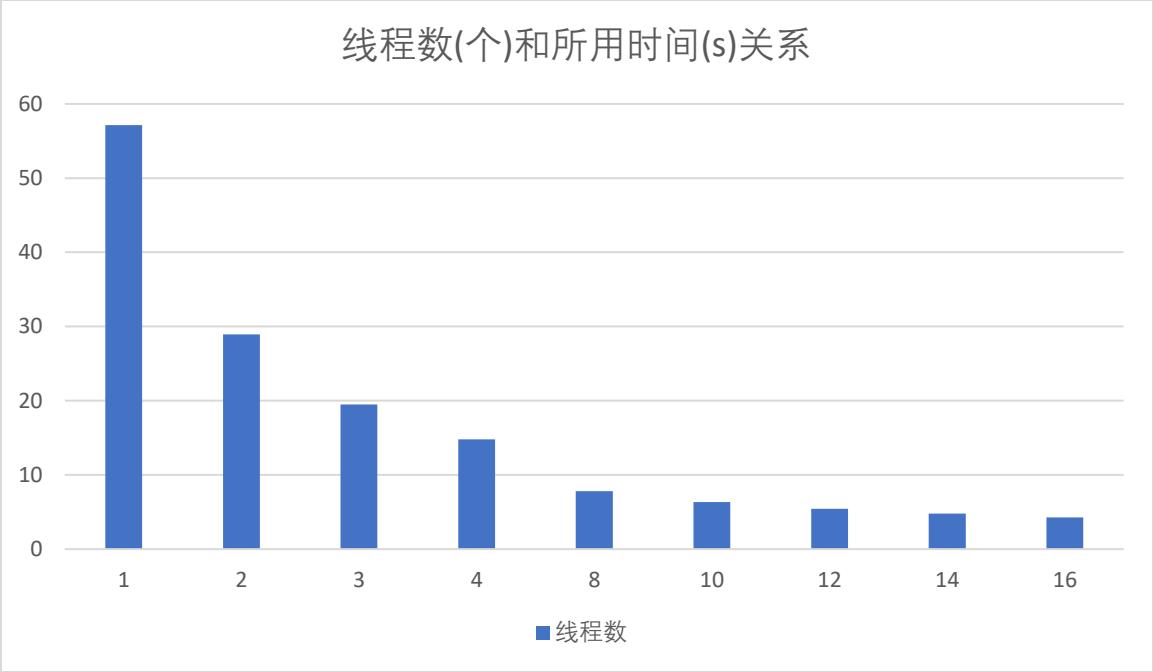
```
g++ -pthread -o sin1.o sin1.c
yhbatch -p thcp1 -n 1 ./sin1.sh
```

```
Using username "tjucic036".
**********************************************************************
* Welcome to NSCC-TJ Supercomputer System.                           *
* For questions or additional information, please contact:           *
* support@nscc-tj.cn (Hardware) / service@nscc-tj.cn (Software)      *
**********************************************************************
Last login: Tue Mar 11 20:38:33 2025 from 10.100.100.3
Disk quotas for grp tjucic_course (gid 10271):
     Filesystem    used   quota   limit   grace   files   quota   limit   grace
         /thfs1  58.33G    512G      1T       -  436725  1000000 2000000       -
tjucic036@ln0:~$ cd lab1
tjucic036@ln0:~/lab1$ g++ -pthread -o sin1.o sin1.c
tjucic036@ln0:~/lab1$ ls
sin1.c  sin1.o  sin1.sh
tjucic036@ln0:~/lab1$ yhbatch -p thcp1 -n 1 ./sin1.sh
Submitted batch job 8209547
tjucic036@ln0:~/lab1$ ls
sin1.c  sin1.log  sin1.o  sin1.sh
```

得到的.log 文件部分内容：

```
===== 1 thread(s) =====
0.841471

real    0m57.138s
user    0m0.153s
sys 0m0.035s

===== 2 thread(s) =====
0.841471

real    0m28.928s
user    0m0.162s
sys 0m0.029s

===== 3 thread(s) =====
0.841471

real    0m19.498s
user    0m0.168s
sys 0m0.021s

===== 4 thread(s) =====
0.841471
```

统计柱状图为：



结论：当 n = 100000 时，对于该任务，随着线程数增加，初期对作业时间的优化较大，当线程数到达某一程度时，优化效果不再明显，可能是互斥锁限制了性能的进一步提升。

七、个人总结

（1）通过实现 Pthread 多线程计算正弦值，掌握了线程创建、任务划分、互斥锁同步基本操作。

（2）了解了对于计算密集型任务，对数据划分有了基本了解，该实验采用循环划分，优点是能有效实现负载均衡，防止有些线程提前结束，造成性能浪费。

（3）实现方式为局部计算，最后全局合并，减少锁操作次数（每个线程先独立计算局部和，最后仅一次加锁），有效降低同步开销。

（4）对于并行计算得到新认识：仅仅增加核数和线程数，不一定时线性的优化结果。

（5）实验中发现：任务规模较小（n=1000）时，多线程加速效果不明显，采用的 n = 100000 是进行多次测试的结果。可能因为线程创建、调度等固定开销在小规模任务中占比较高，增大输入规模后，这种固定开销所占的占比在总时间中的比例降低，可以看到明显的实验现象。

（6）了解了如何使用天河超算，例如，如何在平台上编译程序，提交任务等。