**中山大学计算机院本科生实验报告**

**（2024学年春季学期）**

**课程名称：并行程序设计 批改人：**

|  |  |  |  |
| --- | --- | --- | --- |
| **实验** | **6-Pthreads并行构造** | **专业（方向）** | **计算机科学与技术** |
| **学号** | **21307174** | **姓名** | **刘俊杰** |
| **Email** | **liujj255@mail2.sysu.edu.cn** | **完成日期** | **2024/5/3** |

# **实验目的**

**parallel\_for并行应用**

**使用此前构造的parallel\_for并行结构，将heated\_plate\_openmp改造为基于Pthreads的并行应用。**

**heated plate问题描述：规则网格上的热传导模拟，其具体过程为每次循环中通过对邻域内热量平均模拟热传导过程，即：**

**，**

**其OpenMP实现见课程资料中的heated\_plate\_openmp.c。**

**要求：使用此前构造的parallel\_for并行结构，将heated\_plate\_openmp实现改造为基于Pthreads的并行应用。测试不同线程、调度方式下的程序并行性能，并与原始heated\_plate\_openmp.c实现对比。**

# **实验过程和核心代码**

**2.1 实验思路**

## **①首先使用实验5实现的parallel\_for来模拟实现openmp的加速功能。**

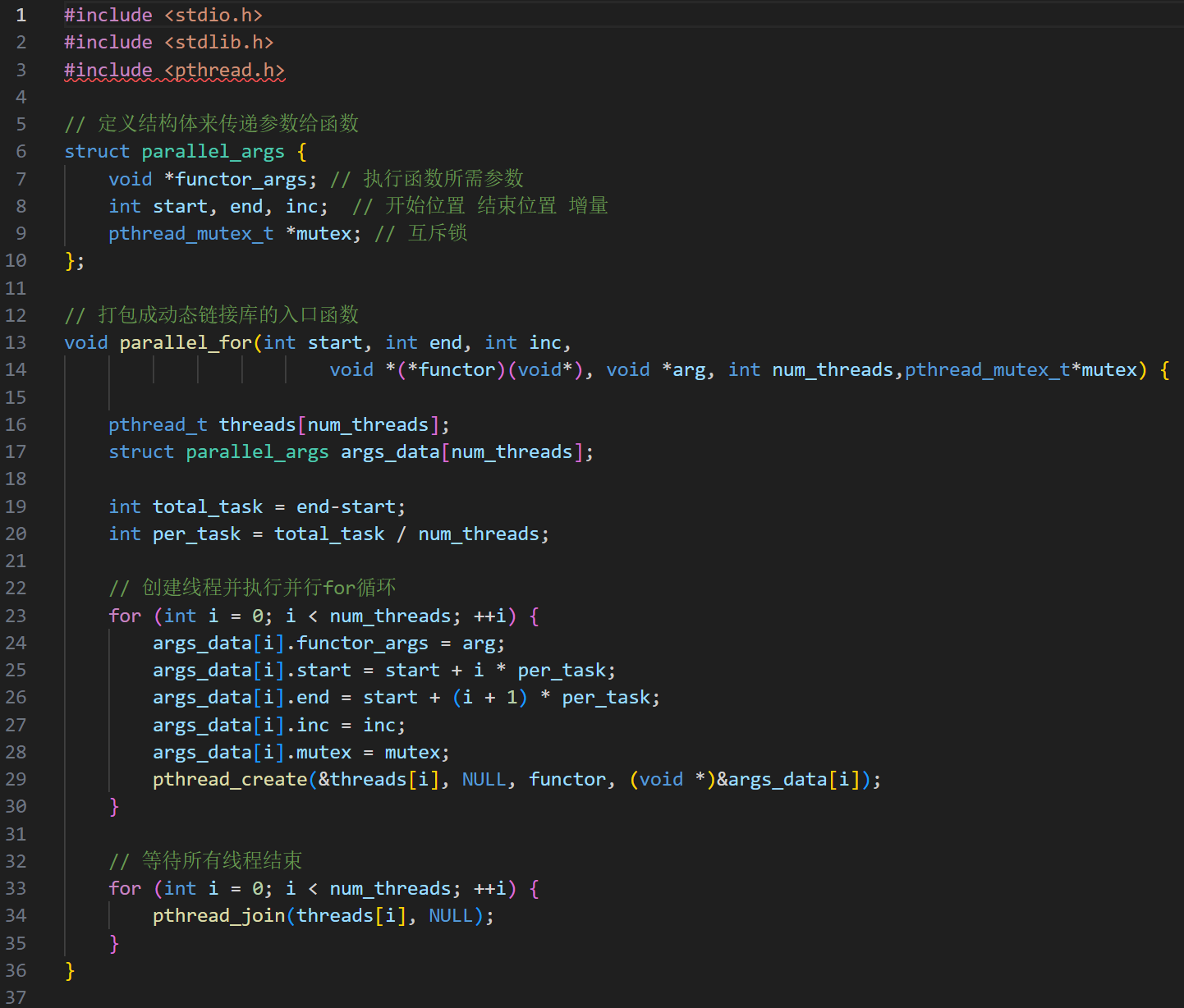
**②将原代码中的代码分为多个部分:，每个部分构造一个函数俩来实现，方便传入到parallel\_for中多线程执行。**

**③按步骤将不同的函数传入到parallel\_for中进行线程加速。**

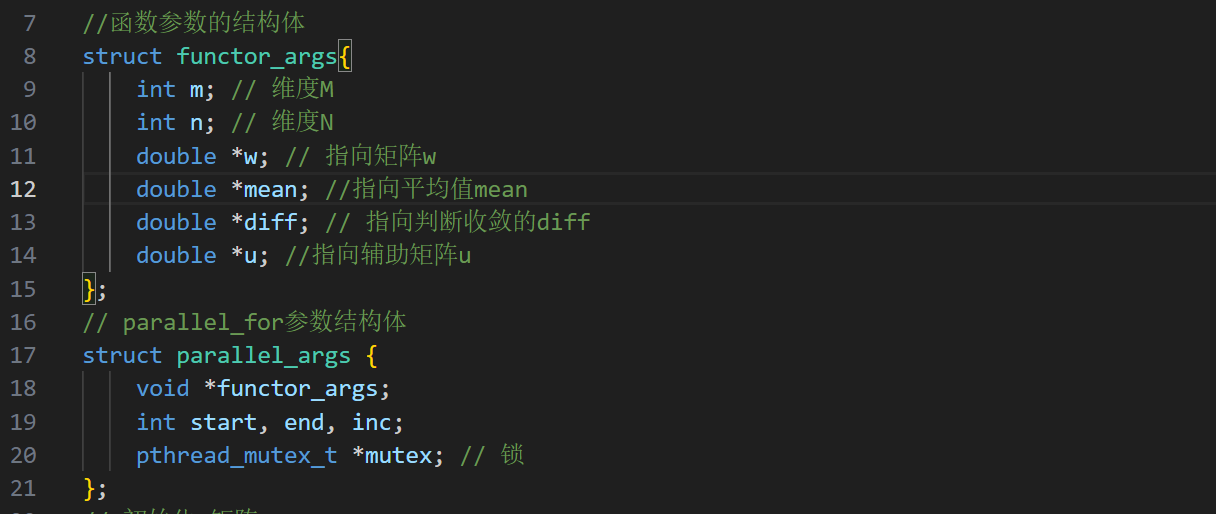
**④输出迭代过程中的diff值和最终收敛需要的迭代次数和运行时间。**

## **2.2 parallel.c**

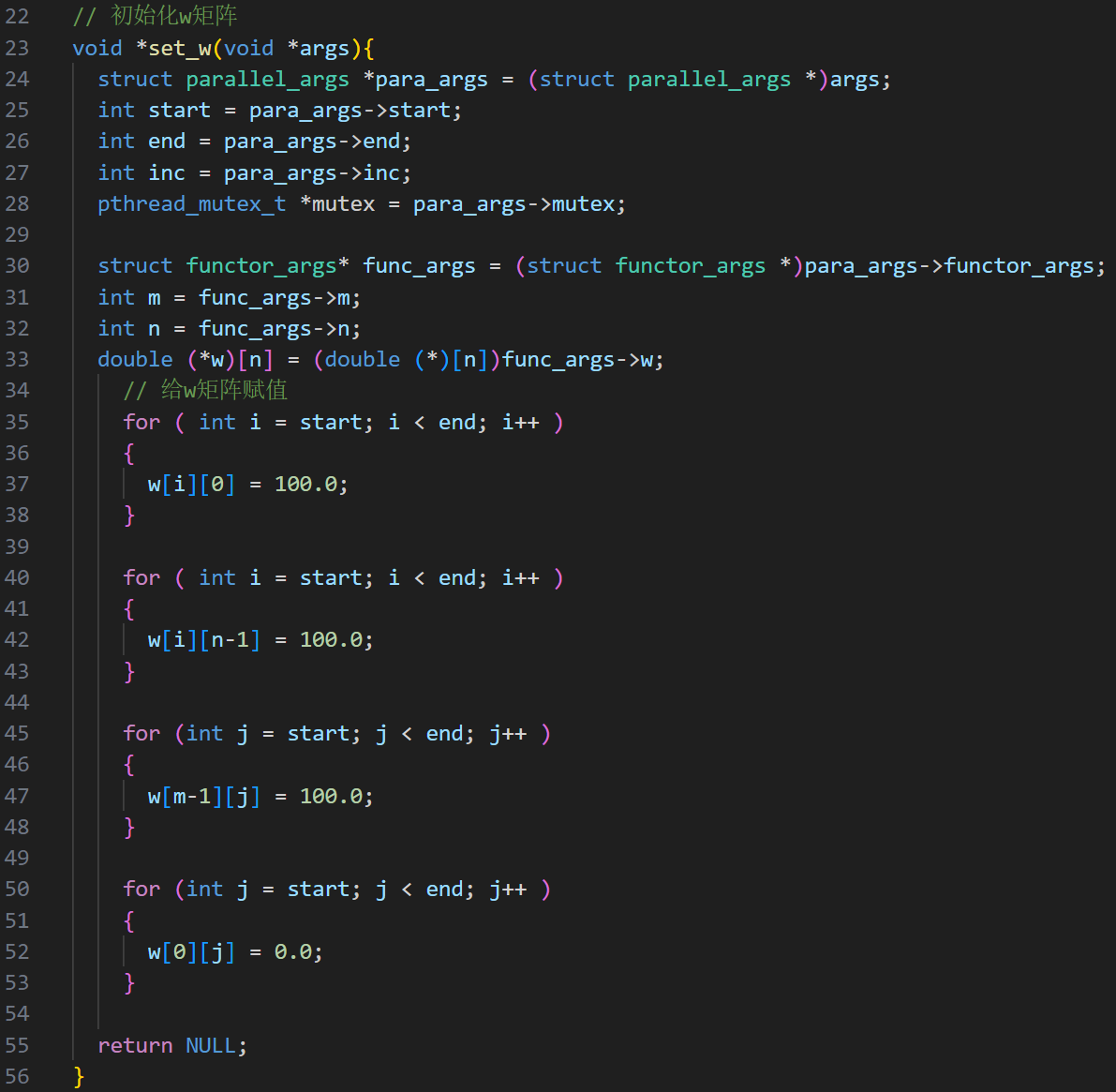
**实验5实现的parallel\_for.c**

****

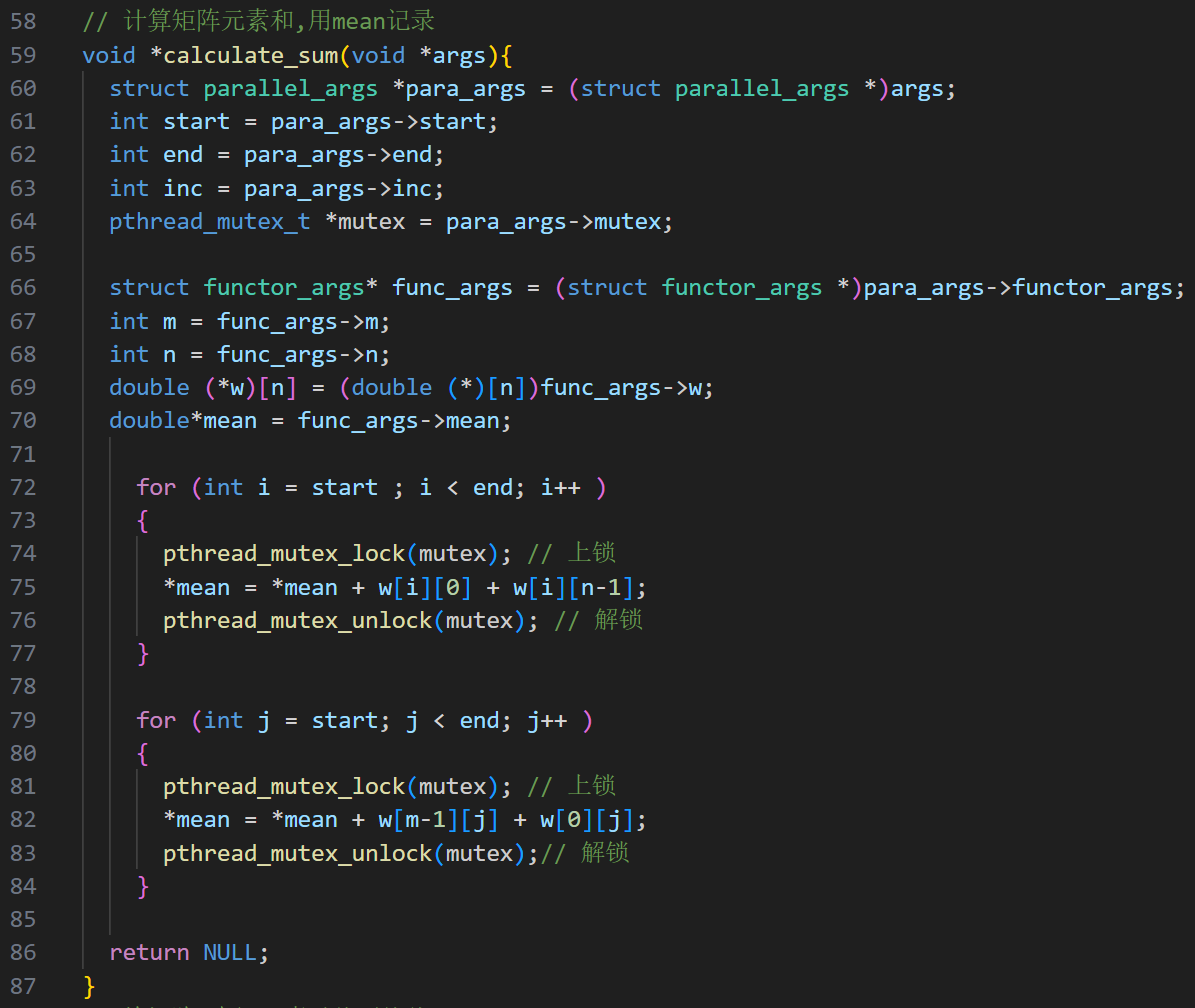
**2.3 pthread版本:heat\_plate\_pthread.c**

**①用到的传参数的结构体**

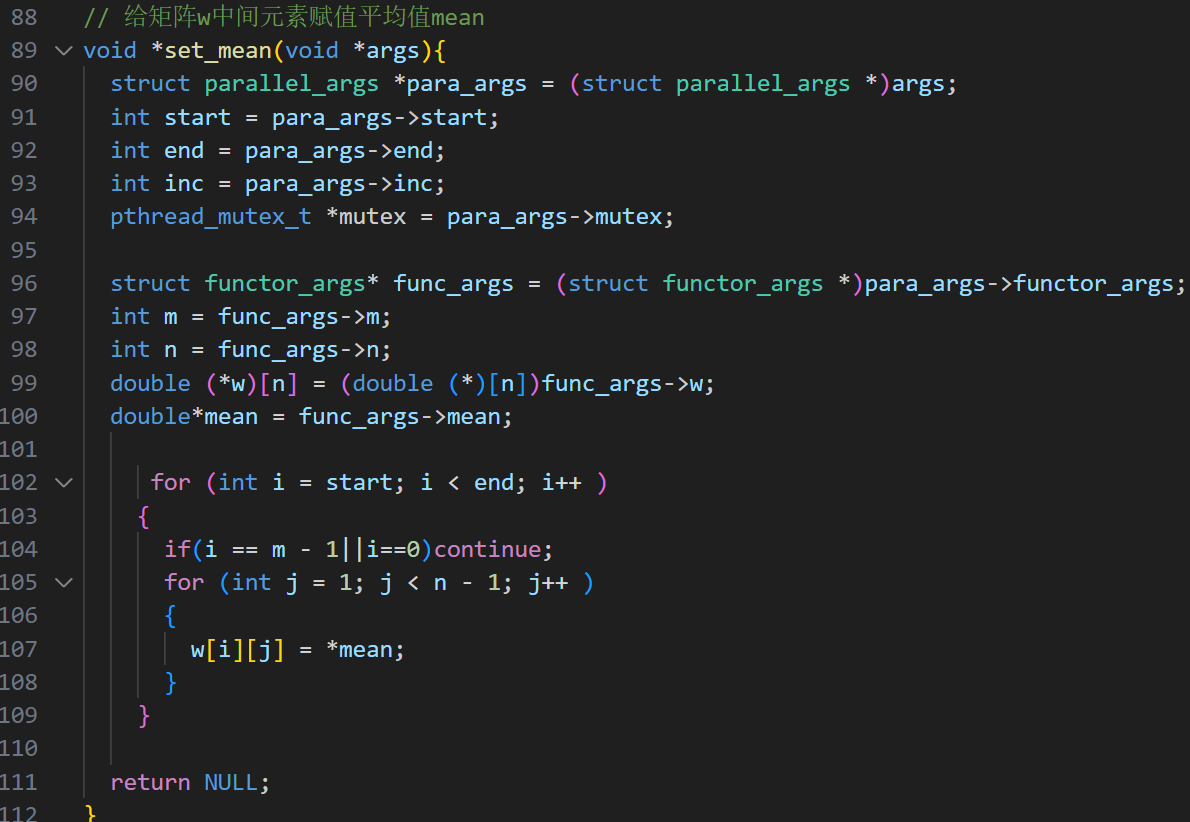
**②初始化矩阵w,使用多线程,每个线程负责相应的位置,给其最左侧、最右侧、最上侧和最底部进行赋值来实现加速。**

****

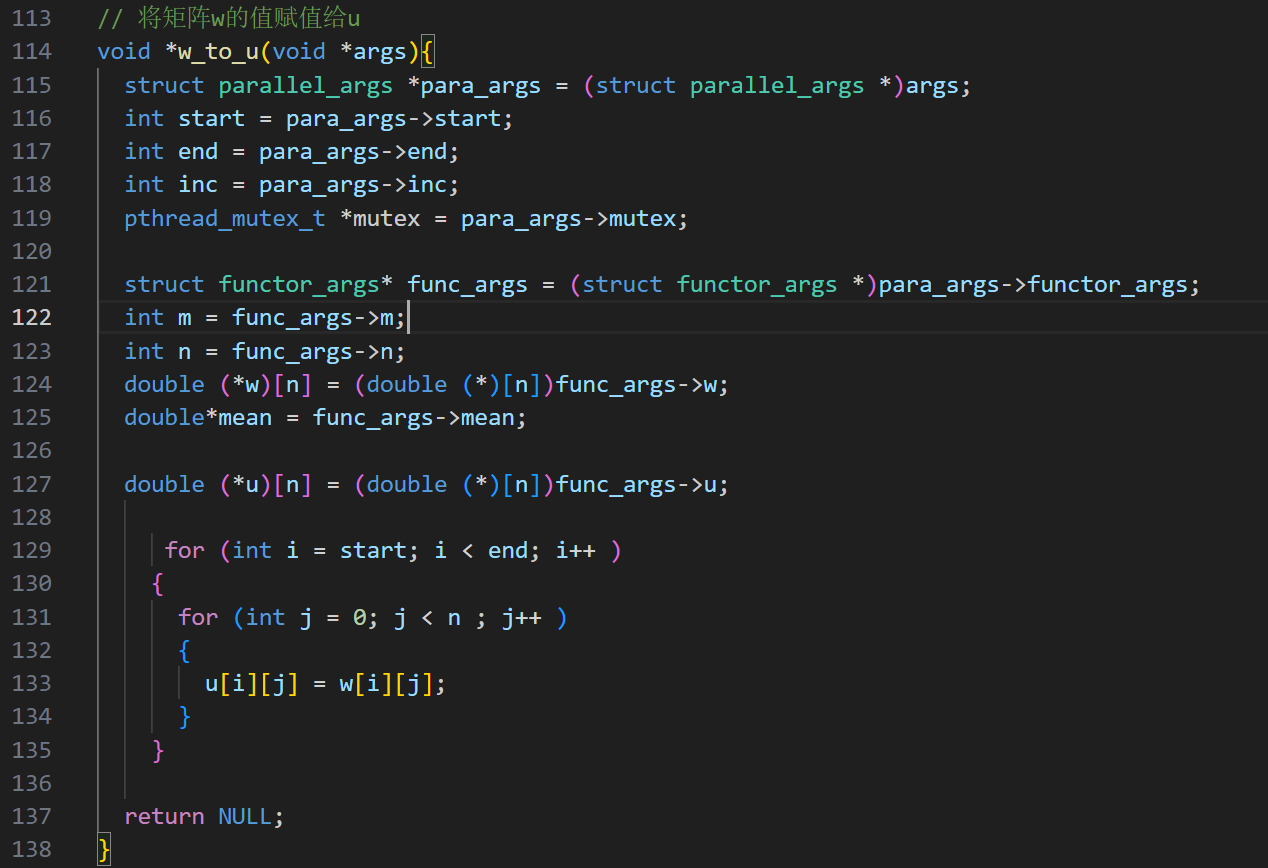
## **③计算初始化后矩阵w中的元素和(方便后续计算出mean)，结果存储到\*mean中(mean指针指向主函数中的mean)，计算过程中需要对互斥区实现加锁解锁操作。**

****

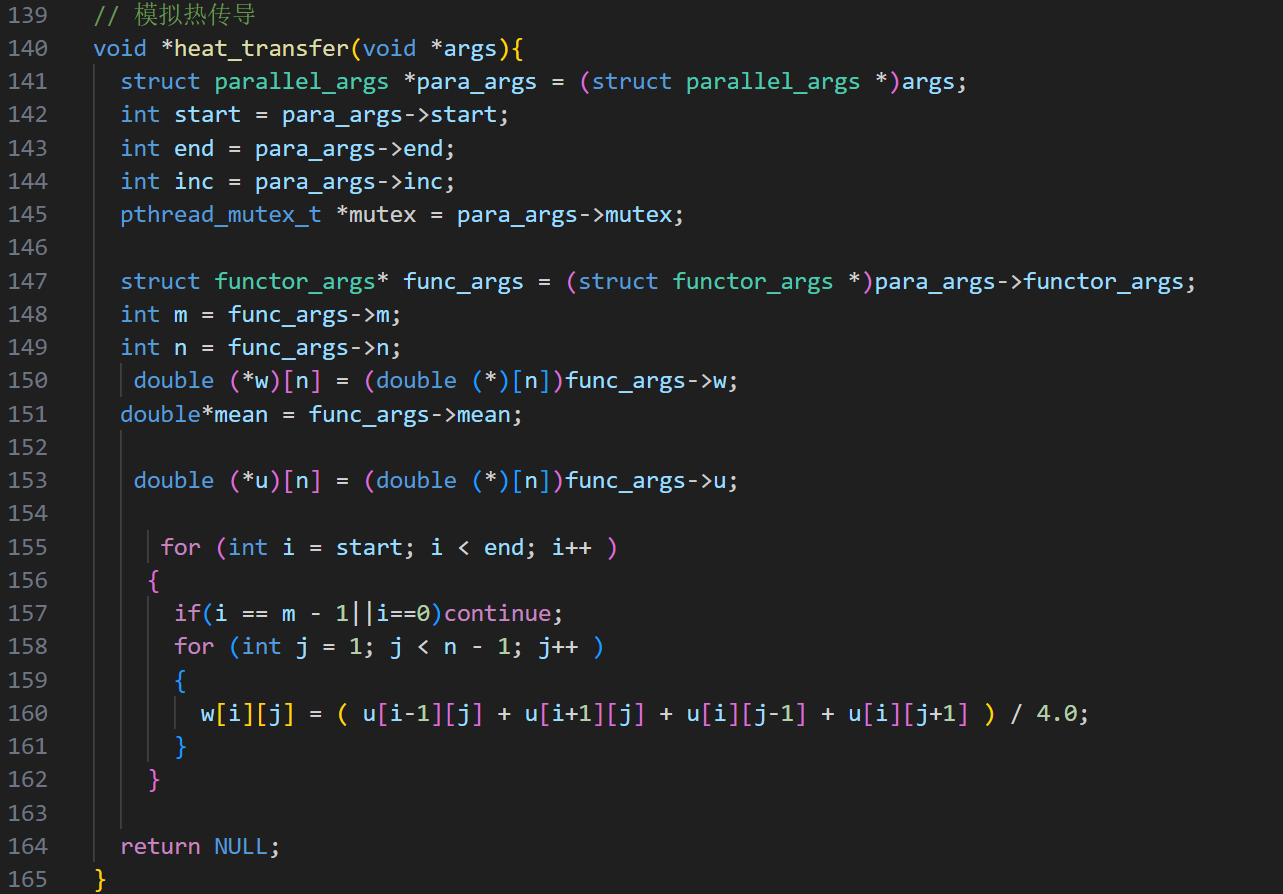
**④计算出mean值后,使用多个线程对网格中间元素赋值mean**

****

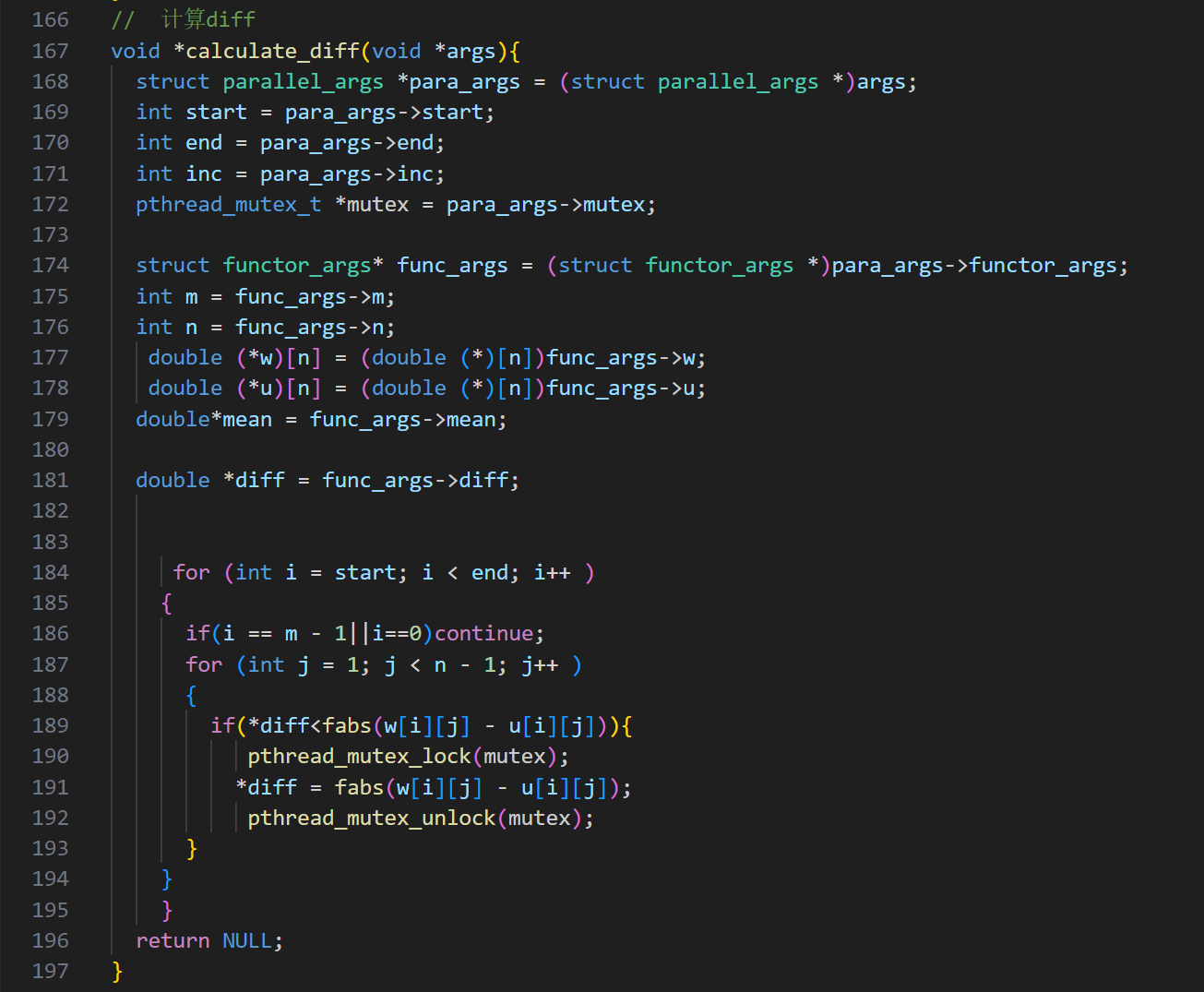
**⑤ 将w的值赋值给辅助矩阵u**

****

**⑥ 多线程模拟网格热传导**

****

**⑦多线程分区域计算diff值,使用锁来实现对diff的互斥访问**

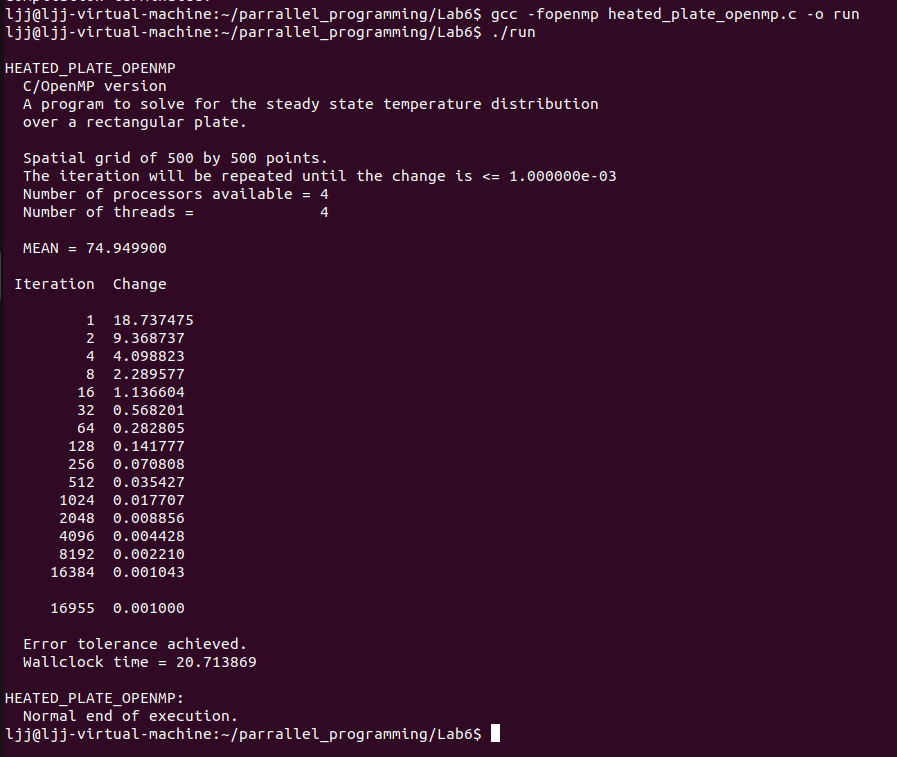
****

# **实验结果**

**3.1 head\_plate\_openmp.c**

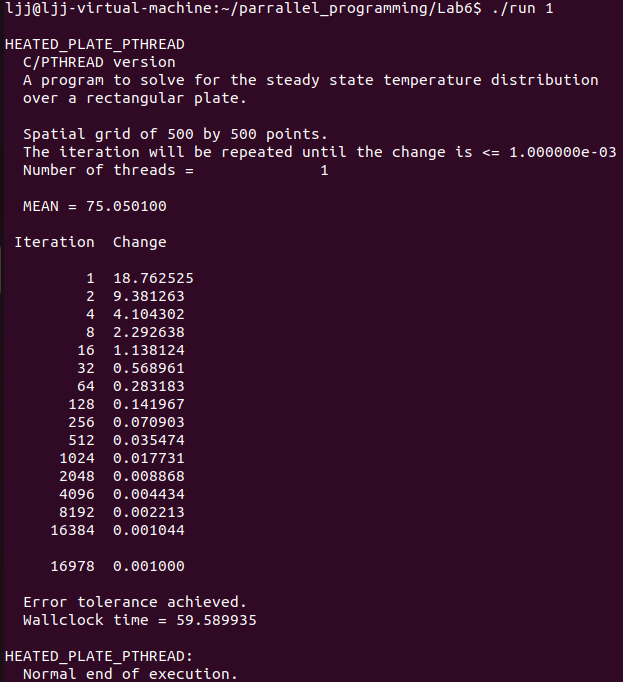
## **openmp版本的实验结果:**

**针对500X500的网格,用4个线程的加速结果:**

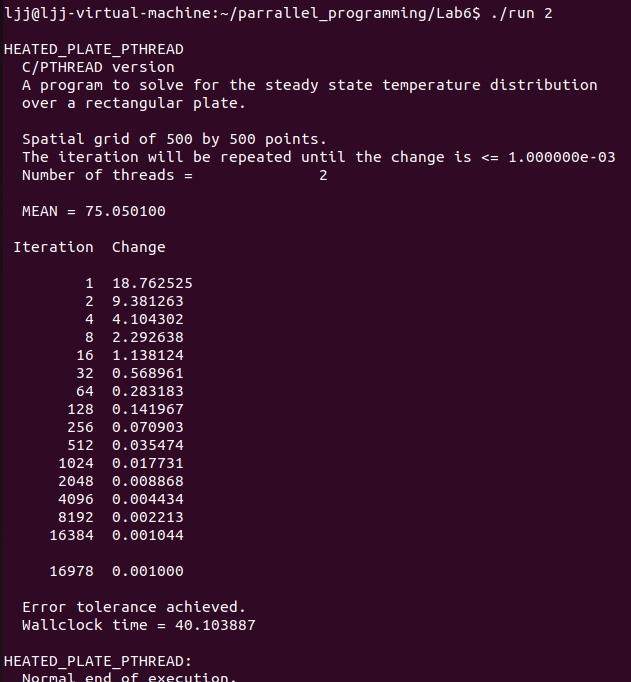
****

## **3.2 parallel\_for实验结果(针对500X500的网格)**

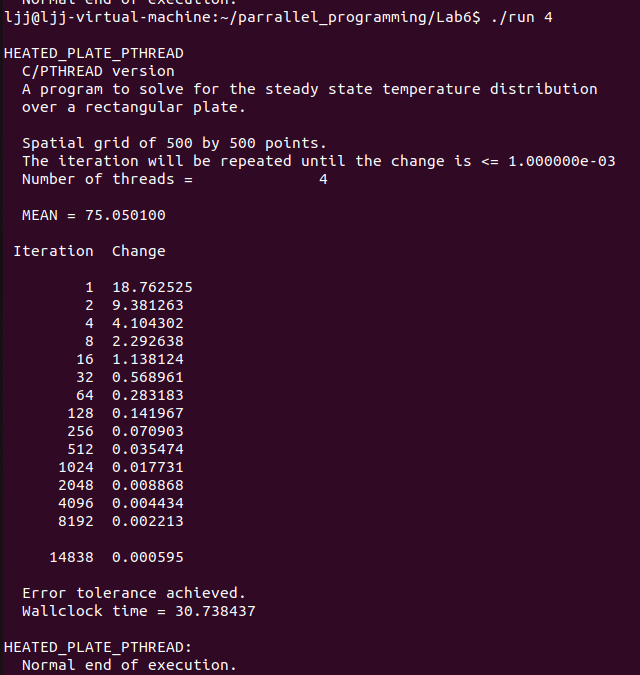
**①1个线程加速:**

****

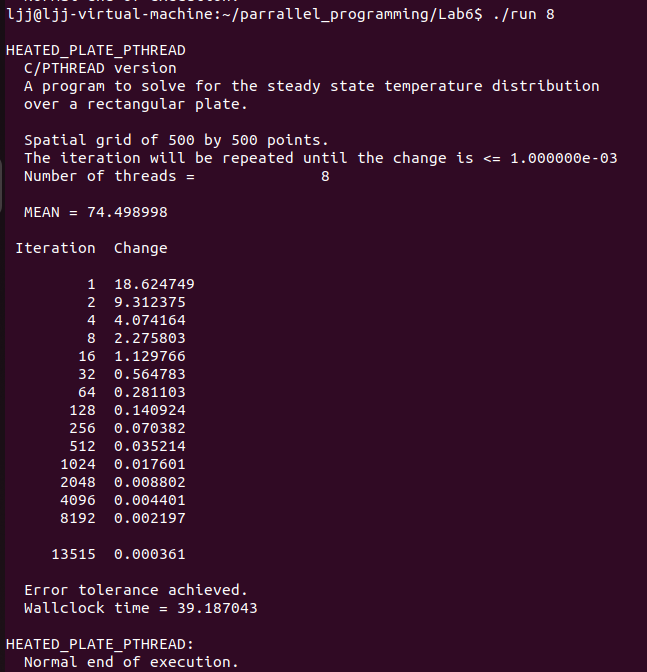
**②2个线程加速:**

****

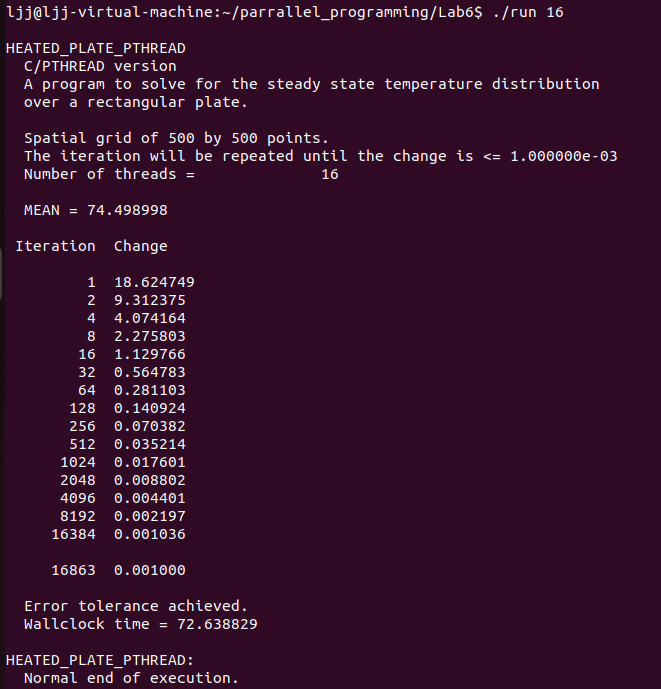
**③4个线程加速:**

****

**④8个线程加速**

****

**⑤16个线程加速**

****

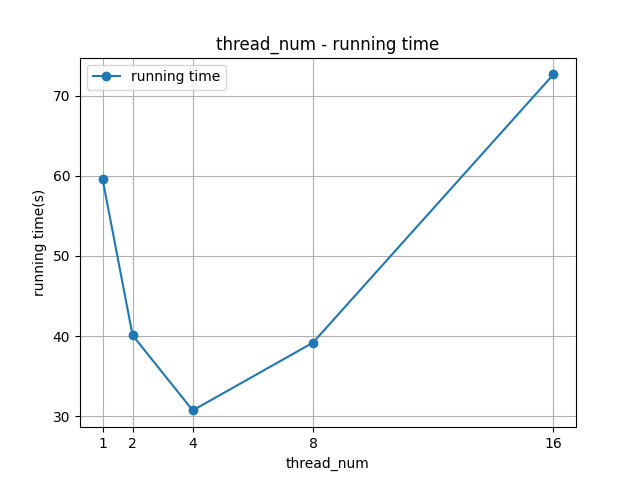
**3.3 综合对比**

**openmp:**

|  |  |  |
| --- | --- | --- |
| **线程数** | **迭代次数** | **运行时间** |
| **4** | **16955** | **20.713869s** |

**parallel\_for:**

|  |  |  |
| --- | --- | --- |
| **线程数** | **迭代次数** | **运行时间** |
| **1** | **16978** | **59.589935s** |
| **2** | **16978** | **40.103887s** |
| **4** | **14838** | **30.738437s** |
| **8** | **13515** | **39.187043s** |
| **16** | **16863** | **72.638829s** |

****

**由于虚拟机资源数有限,不足以支持8线程和16线程的运行,所以其运行时间反而增长了。**

**看从1个线程到4个线程的运行时间的减少,可以看出parallel\_for成功实现了对热传导模拟的多线程加速。**

**但是与openmp的4个线程加速的效果来看，parallel\_for的效果相对来说没有那么好,应该是parallel\_for只有静态调度，没有实现其他的线程调度,可能导致有部分线程资源浪费，同时openmp库中可能有一些优化操作。**

# **实验感想**

**4.1问题**

**①程序运行过程中出现段错误**

**原因及解决办法：**

**自己存储函数参数的结构体是指针，并没有为其分配内存空间。**

**最后为其开辟空间后解决。**

**②出现线程越多,运行时间越长的结果**

**原因及解决办法：**

**出现这种情况的原因是因为我在一开始的计时过程中使用了clock函数计时。**

**clock()函数的功能: 这个函数返回从“开启这个程序进程”到“程序中调用C++ clock()函数”时之间的CPU时钟计时单元（clock tick）数当程序单线程或者单核心机器运行时，这种时间的统计方法是正确的。但是如果要执行的代码多个线程并发执行时就会出问题，因为最终end-begin将会是多个核心总共执行的时钟嘀嗒数，因此造成时间偏大。**

**最后使用clock\_gettime()来计时,解决了这个问题。**

**4.2 感悟**

**学会了使用自己构造的parallel\_for.c对特定的问题进行加速，同时在不同问题中注意到了对共享变量的保护，会使用锁来形成互斥去来避免竞争访问。**

**同时注意到了多线程程序中使用clock计时的不可靠性,因为clock中的时钟会受到多个线程核心的影响而造成时间增加。**

**之后的改进可以为parallel\_for.c实现可选择的不同的线程调度方式(如实现一个线程池来实现动态调度,给每个空闲的线程分配一个任务,避免有的线程执行完自身的任务而闲置造成资源浪费)，避免一些线程资源的浪费来进一步实现加速。**