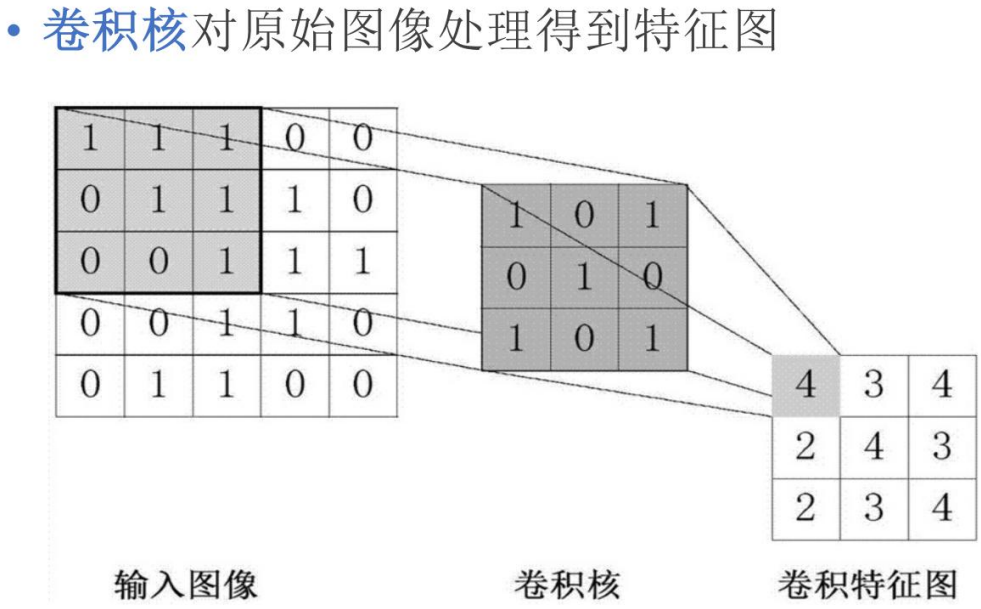
《并行计算》实验报告（正文）

姓名 刘恒星 学名 2022229044 完成时间 2023-4-6

一、实验名称与内容

实验名称：多线程计算卷积

实验内容：卷积是一种积分变换的数学方法，广泛应用于通信、物理、图像处理等领域。图像处理中，卷积操作就是卷积核（过滤器 / Filter）在原始图像中进行滑动得到特征图的过程，如图所示。



二、实验环境的配置参数

CPU：国产自主 FT2000+@2.30GHz 56cores

节点数：5000

内存：128GB

网络：天河自主高速互联网络 400Gb/s

单核理论性能（双精度）：9.2GFlops

单节点理论性能（双精度）：588.8GFlops

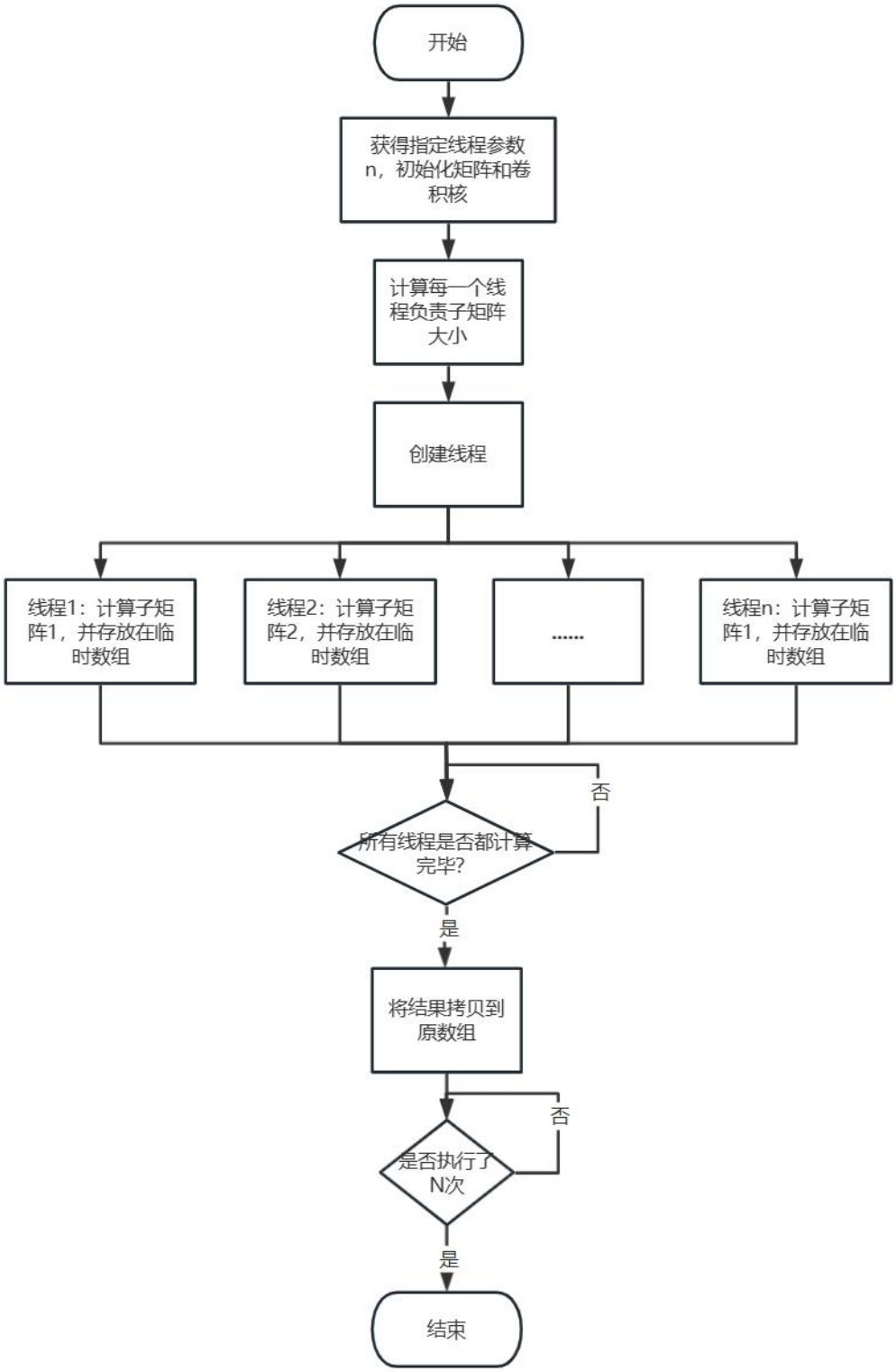
二、实验题目问题分析

该题目是一个计算矩阵卷积的问题，此问题中，需要卷积核遍历矩阵进行计算。可以抽象为遍历数据域计算最后整合的问题。

对于遍历数据域计算最后进行整合这种类型的问题，我们可以通过划分数据域进行并行优化。具体来说，我们可以将矩阵划分为子矩阵，每一个子矩阵用一个卷积核进行计算，讲子矩阵的结果保存在临时数组中，最后等待所有线程计算完毕，将数据从临时数据拷贝到原矩阵中，从而达到多线程并行优化的效果。

四、方案设计

流程图如下：


开始
获得指定线程参数n，初始化矩阵和卷积核
计算每一个线程负责子矩阵大小
创建线程
线程1：计算子矩阵1，并存放在临时数组
线程2：计算子矩阵2，并存放在临时数组
.....
线程n：计算子矩阵1，并存放在临时数组
所有线程是否都计算完毕?
否
是
将结果拷贝到原数组
是否执行了N次
否
是
结束

伪代码如下:

```
conv2d(id):
    start_row = id * per_thread_rows;
    end_row = (id + 1) * per_thread_rows;
    result[][];

    for iter from 0 to N:
        for i from 0 to per_thread_rows:
            for j from 0 to MAXN - ks + 1:
                sum = 0;
                for ki from 0 to ks:
                    for kj from 0 to ks:
                        sum += filter[ki][kj] * img[i + ki + start_row][j + kj];
                result[i][j] = sum;
        pthread_barrier_wait(&barrier);
        for i from 0 to per_thread_rows:
            for j from 0 to MAXN - ks + 1:
                img[i+1+start_row][j+1] = result[i][j];
        pthread_barrier_wait(&barrier);

for i from 0 to thread_num:
    ind[i] = i;
    pthread_create(&tid[i], NULL, conv2d, (void *)&(ind[i]));

for i from 0 to thread_num:
    pthread_join(tid[i], NULL);
```

五、实现方法

首先，在程序中定义好矩阵的大小，本次实验定义矩阵原始大小为 256*256，在 padding 之后大小为 258*258，卷积核大小为 3*3。初始化函数中为原始矩阵中间 256*256 的内容填充随机数，卷积核采用的是经典的边缘提取卷积核

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

。随后从参数中获取线程数，并计算好子矩阵大小。设 per_thread_row = 256/thread_num，那么每一个线程负责的子矩阵大小为 per_thread_row * 256。

随后进行卷积运算，线程通过函数参数得到 id 号，从而计算出自己的子矩阵在原始矩阵的起始位置。开辟一个 per_thread_row * 256 大小的临时数组来记录运算结果。为了防止先计算完成的线程干扰后还在计算的线程，需要等待至所有线程计算完毕之后同意复制结果到原数组。这里使用 pthread_barrier_wait 函数来同步线程。

复制完毕之后，需要用 pthread_barrier_wait 函数等到所有线程都复制完毕，才能进行下一次的计算。

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）

本次实验矩阵大小为 256*256，padding 后为 258*258，一共计算 3 次，实验结果保证正确，实验结果如下：

串行程序 运行时间：0.016471

2 线程数并行程序 运行时间：0.008388 加速比：1.91702 效率：0.958512

4 线程数并行程序 运行时间：0.004422 加速比：3.63636 效率：0.909091

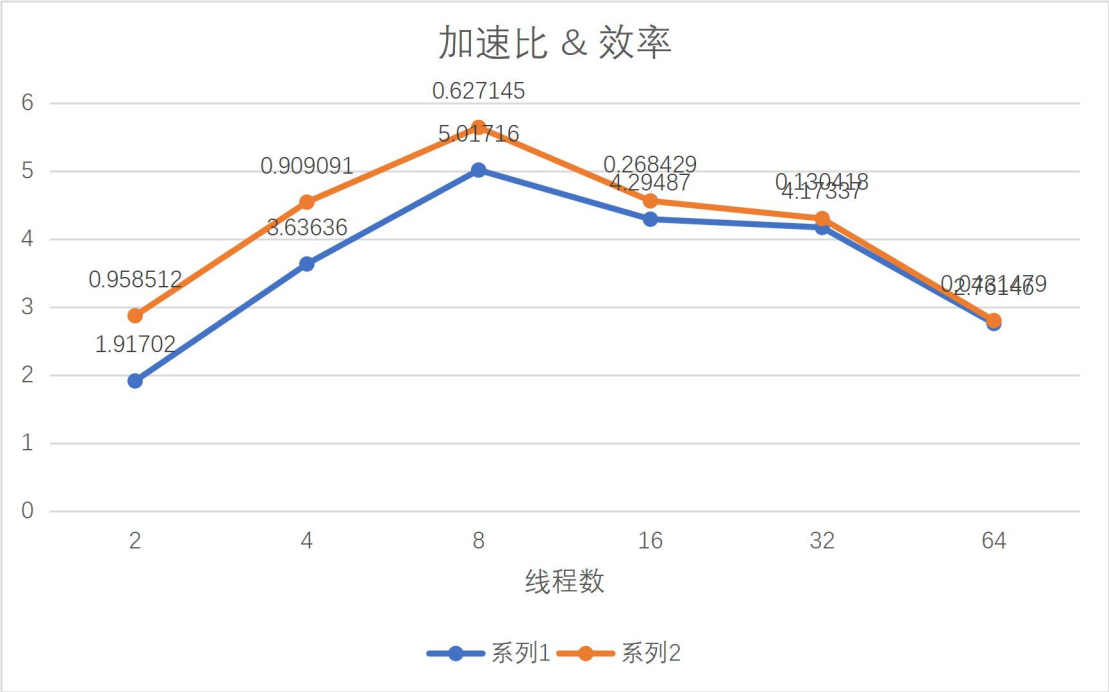
8 线程数并行程序 运行时间：0.003205 加速比：5.01716 效率：0.627145

16 线程数并行程序 运行时间：0.003744 加速比：4.29487 效率：0.268429

32 线程数并行程序 运行时间：0.003853 加速比：4.17337 效率：0.130418

64 线程数并行程序 运行时间：0.005823 加速比：2.76146 效率：0.043147

加速比和效率曲线如下



可以发现，随着线程数的增加，运行速度持续减少，加速比和效率都是先增后减，原因是过多的线程会额外引入开销，当额外开销过多，效率和加速比就会降低。

七、个人总结

通过这次实验，明白了如何使用 pthread 库实现多线程编程，了解了并行程序设计。从这次实验遇到的困难集中在如何设计并行优化上，一开始并不会处理这个问题里的线程同步问题。这次实验也让我明白了 pthread 库的进一步使用和一些较复杂的并行程序的设计。通过实验结果，可以发现线程并不是越多越好，线程的增加会引起效率的降低，不加思考的引入线程会导致额外的开销，如何在效率和加速比中得到权衡是一个值得思考的问题。