《并行计算》实验报告（正文）

姓名 刘恒星 学名 2022229044 完成时间 2023-04-18

一、实验名称与内容

实验四：多进程计算卷积

本实验针对实验二问题，采用 MPI+OpenMP 编程模型实现卷积计算。节点间采用 MPI，节点内采用 OpenMP。需要制定多层划分策略。

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

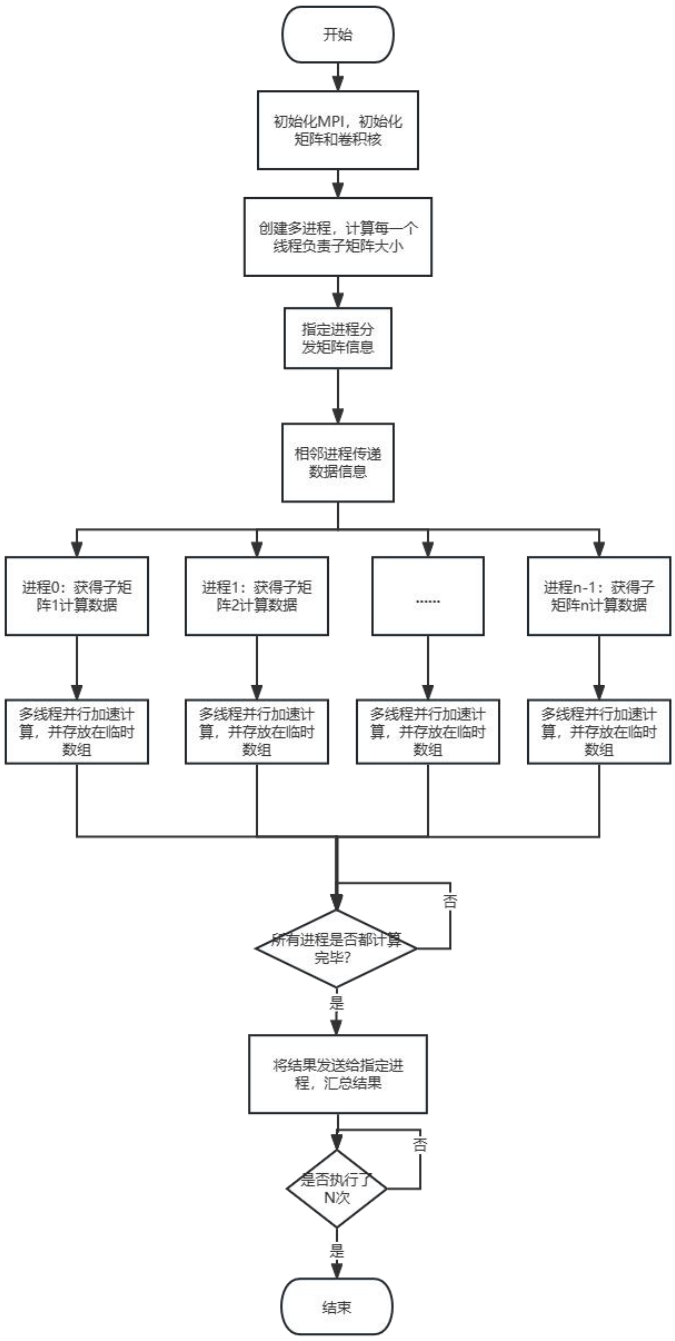
单节点理论性能（双精度）：588.8GFlops

三、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，指定一个进程发送矩阵信息，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，卷积的计算使用多线程并行加速，子矩阵的结果保存在临时数组中，最后等待所有进程计算完毕，将数据从临时数据拷贝到原矩阵中，从而达到并行优化的效果。

四、方案设计




```
conv2d(int** img, int **result, int row, int col, bool last)
{
    #pragma omp parallel for num_threads(t)
    Result = CalculateConv2d(filter, img);
}


    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    row_per_process = MAXN / size;
```

```
    id = my_rank;
    // if it is the first time run this program, img[][] need to be init
    if(id == root && need_init)
    {
        Init(img);
    }
    for(iter = 0; iter < N; iter++)
    {
        MPI_Bcast(&img[0][0], MAXN*MAXN, MPI_INT, root, MPI_COMM_WORLD); // root process Bcast the
content of img
        /*
        * **a is a temp array to save data from st_row to ed_row in order to help calculate conv2d
        */
        a[][] = None;
        st_row = id * row_per_process;
        ed_row = st_row + row;
        Copy(a, img);
        /*
        * calculate conv2d need data of other process, use MPI_Sendrecv to trans the data which
needed
        */
        send_to = id - 1;
        receive_from = id + 1;
        if(id == 0)
        {
            send_to = MPI_PROC_NULL;
        }
        if(id == size - 1)
        {
            receive_from = MPI_PROC_NULL;
        }
        tag1 = 1;
        MPI_Sendrecv(a[0], MAXN, MPI_INT, send_to, tag1, a[row], MAXN, MPI_INT, receive_from, tag1,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Sendrecv(a[1], MAXN, MPI_INT, send_to, tag1, a[row+1], MAXN, MPI_INT, receive_from,
tag1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        /**
         * res[][] to save the result of conv2d from st_row to ed_row
        */
        res[][];
        conv2d(a, res, row, MAXN, id==size-1);
        MPI_Barrier(MPI_COMM_WORLD); // waiting for all process
        MPI_Gather(res[0], row*MAXN, MPI_INT, img[0], row*MAXN, MPI_INT, root, MPI_COMM_WORLD);
```

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 2048*2048，卷积核大小为 3*3。初始化函数中为原始矩阵中间 2048*2048 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

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

。随后用 MPI_Init 初始化多进程环境，调用函数获得参数中指定进程数数，并计算好子矩阵大小。设 per_process_row = 2048/process_num，那么每一个进程负责的子矩阵大小为 per_process_row * 2048。

因为卷积运算在边缘的时候需要相邻进程数据的帮助，考虑到卷积核大小是 3*3，所以我们需要将下面进程的数据传给上面进程，用 MPI_Sendrecv 向相邻进程发送数据。

随后进行卷积运算，进程通过函数 MPI_Comm_Rank 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_process_row * 2048 大小的临时数组来记录运算结果。其中卷积的运算需要用 OpenMP 的制导语句进行多线程加速。为了防止先计算完成的进程干扰后还在计算的进程，需要等待至所有进程计算完毕之后统一复制。这里使用 MPI_Barrier 函数来同步进程。

计算完毕之后，使用 MPI_Gather 函数来将各个继承计算结果汇总的 root 进程下，由 root 进程管理最后的结果。汇总完毕之后，才能进行下一次的计算。

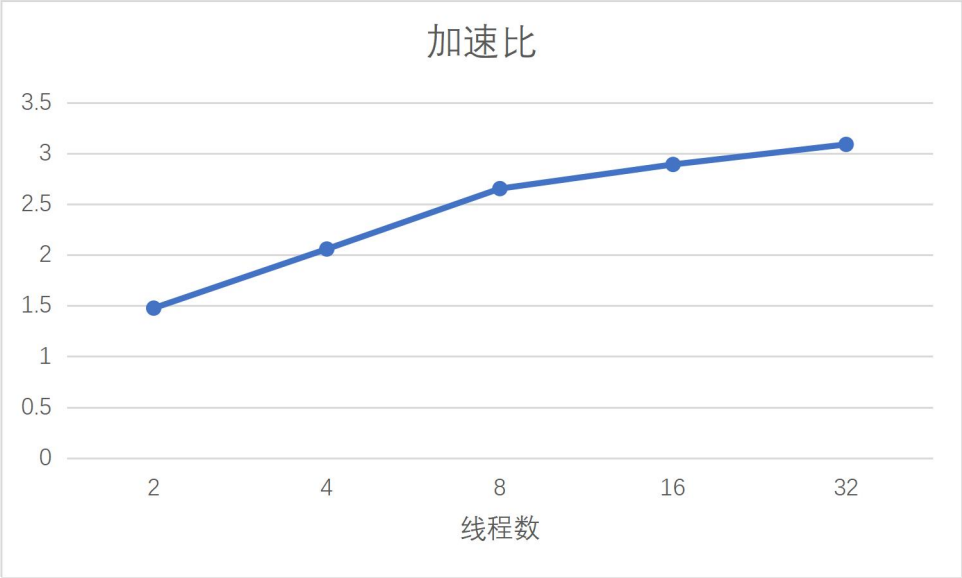
六、结果分析

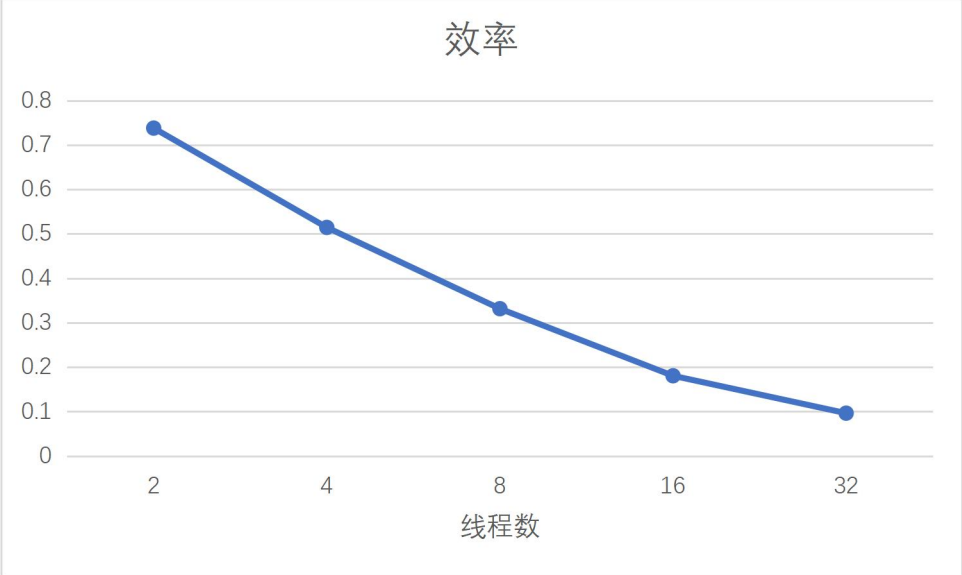
经过测试，保证代码结果的正确性，以下是实验结果

串行运行时间：4.80116

单进程多线程结果：

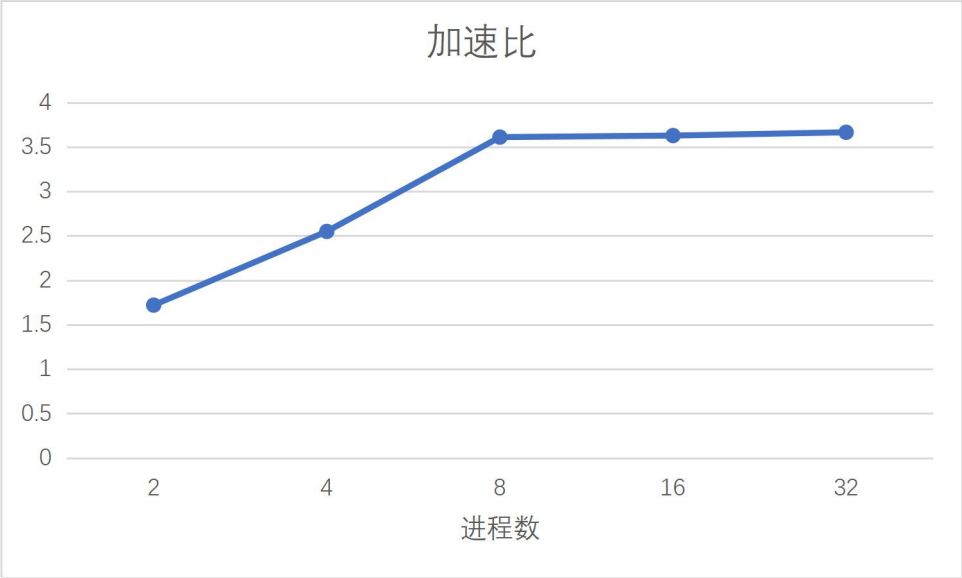
进程数	线程数	运行时间	加速比	效率
1	2	3.25352	1.475681723	0.737840862
1	4	2.33343	2.057554758	0.51438869
1	8	1.81024	2.65222291	0.331527864
1	16	1.66132	2.889967014	0.180622938
1	32	1.55514	3.087284746	0.096477648

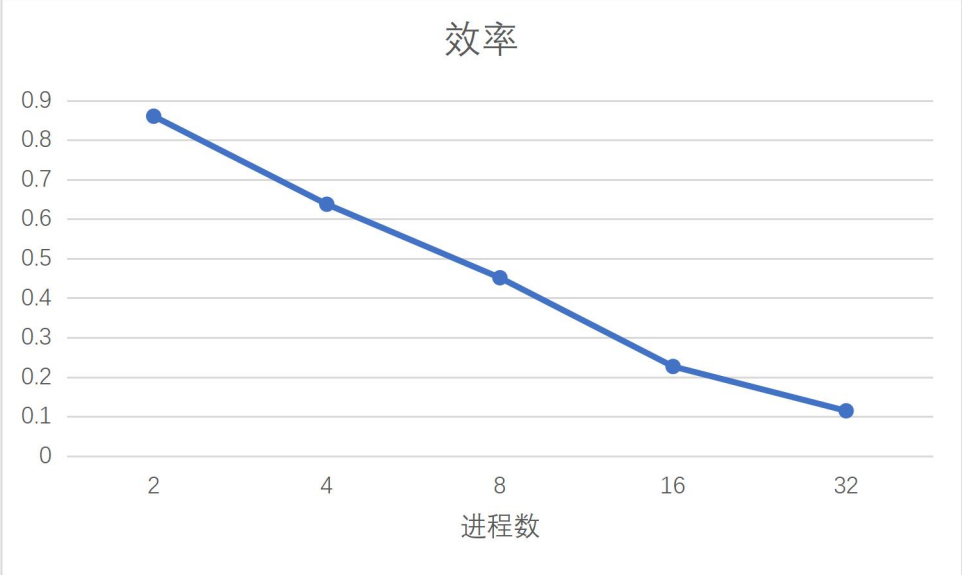




多进程单线程实验结果:

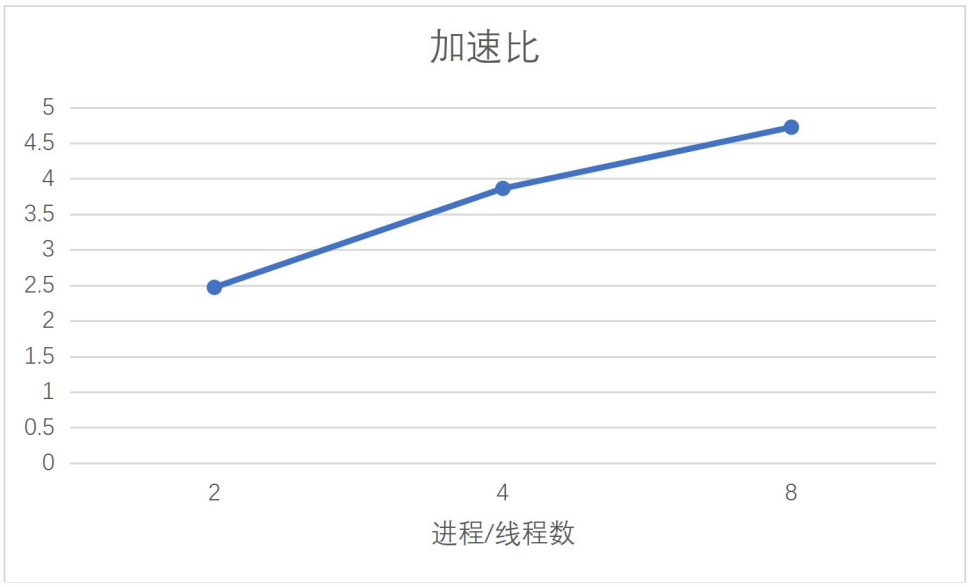
进程数	线程数	运行时间	加速比	效率
2	1	2.79078	1.720364916	0.860182458
4	1	1.88323	2.549428376	0.637357094
8	1	1.3298	3.61043766	0.451304707
16	1	1.32345	3.627760777	0.226735049
32	1	1.31011	3.664699911	0.114521872

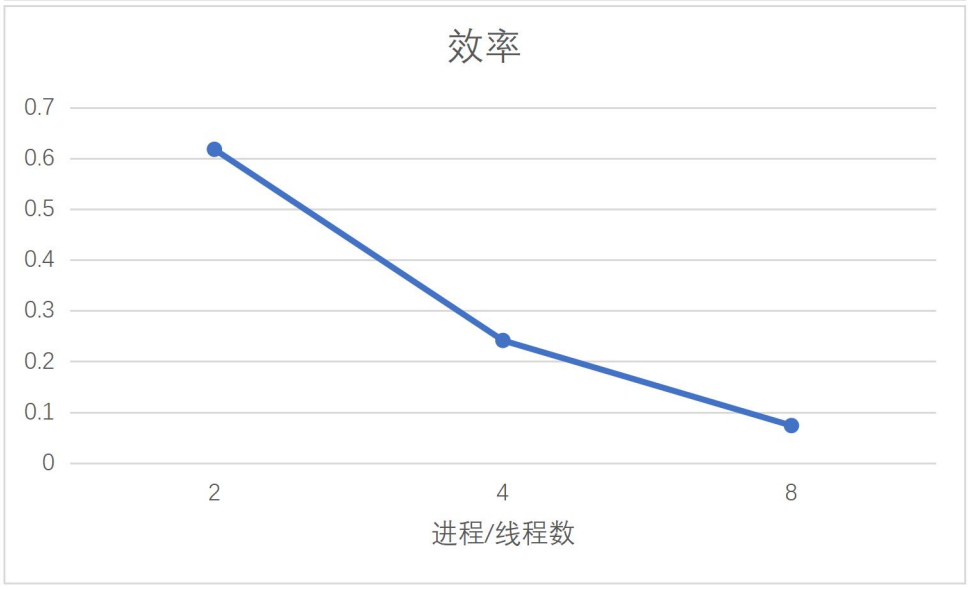




多进程多线程实验结果:

进程数	线程数	运行时间	加速比	效率
2	2	1.94328	2.470647565	0.617661891
4	4	1.24331	3.861595258	0.241349704
8	8	1.01654	4.723040903	0.073797514





可以发现，在使用多进程多线程的时候，运行速度远比单进程或者单线程要快很多，但是同时随着进程和线程的数量增多，效率也在迅速下滑。

七、个人总结

通过这次实验，明白了如何使用 MPI 库实现多进程编程，并学会了 OpenMP+MPI 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。这次实验也让我明白了 MPI 和 OpenMP 一起使用和一些较复杂的并行程序的设计。通过实验结果，可以看出处理器越多，时间越快，但是效率越低。如何在效率和加速比中得到权衡是一个值得思考的问题。