《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

实验三：多进程计算卷积

采用 MPI 编程模型实现卷积计算。

划分方法可参考课程中的 Jacobi 迭代，将原始图像划分成 p（进程数）个子块，每个进程处理一个子块，进行 N 次卷积计算，计算中每一个进程都要向相邻的进程发送数据，同时从相邻的进程接收数据

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

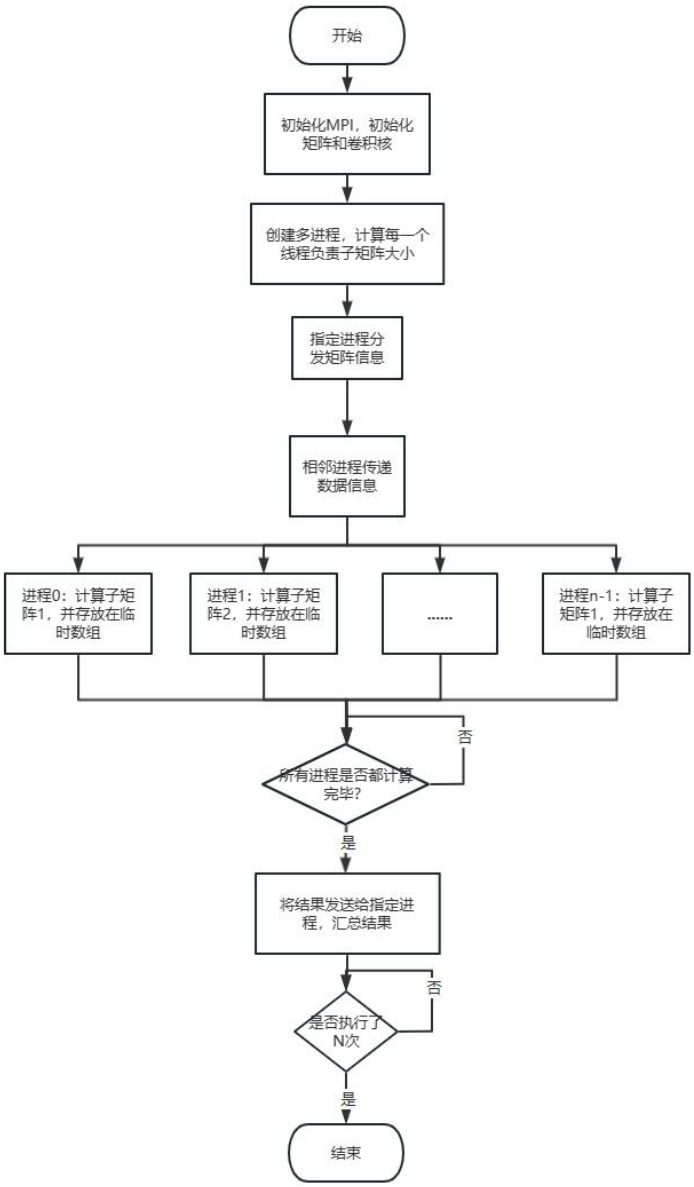
单节点理论性能（双精度）：588.8GFlops

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，讲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




```
    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。

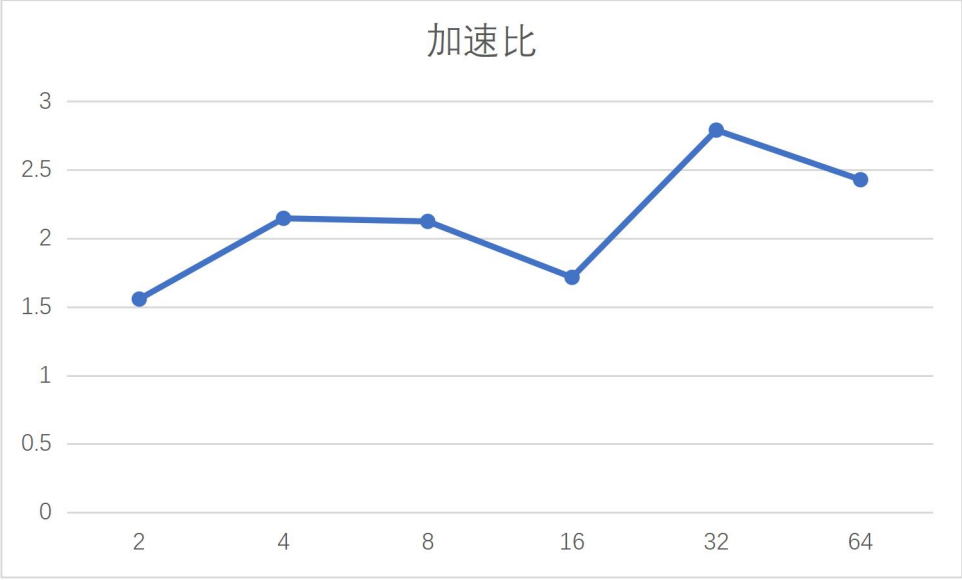
随后进行卷积运算，进程通过函数 MPI_Comm_Rank 得到 id 号，从而计算出自己的子矩阵在原始矩阵的起始位置。开辟一个 per_process_row * 2048 大小的临时数组来记录运算结果。为了防止先计算完成的进程干扰后还在计算的进程，需要等待至所有进程计算完毕之后统一复制。这里使用 MPI_Barrier 函数来同步进程。

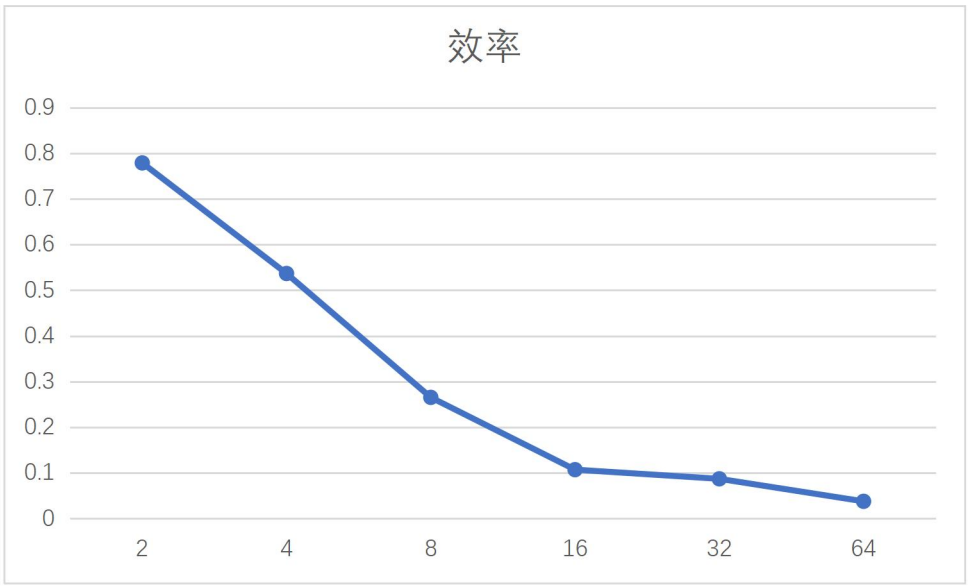
计算完毕之后，使用 MPI_Gather 函数来将各个继承计算结果汇总的 root 进程下，由 root 进程管理最后的结果。汇总完毕之后，才能进行下一次的计算。

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

进程数	运行时间	加速比	效率
串行	4.91033		
2	3.15308	1.557312215	0.778656108
4	2.28764	2.146460982	0.536615245
8	2.31204	2.123808412	0.265476051
16	2.86224	1.71555495	0.107222184
32	1.76005	2.789880969	0.08718378
64	2.02232	2.428067764	0.037938559





可以发现，随着进程数的增加，运行时间大体呈现减下趋势，加速比上升，但是效率也会降低。导致这个问题的原因是因为过多的进程创建和销毁会引入额外的开销，而且进程之间的消息传递也会带来一定的性能负担。

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，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进程同步问题，还有一开始没有非常明白 MPI 背后的原理，不知道 MPI 背后的资源是独立的，需要用进程通信去同步信息。这次实验也让我明白了 MPI 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，可以发现进程并不是越多越好，进程的增加会引起效率的降低，不加思考的引入进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。