中山大学计算机院本科生实验报告

(2023 学年春季学期)

课程名称：超级计算原理与实践　　批改人：

实验	期末大作业（并行卷积）	专业（方向）	计算机科学与技术（人工智能与大数据）
学号	21307181	姓名	杨子昂
Email	Yangzang3@mail2.sysu.edu.cn	完成日期	2023.6.18

1. 实验目的（200 字以内）

本次大作业具体分为两个实验内容，第一个是利用 MPI+OpenMP 实现卷积操作，第二个是 im2col 方法实现卷积操作。并且将输入规模从 32 开始提升一直到 4096，统计运算时间。

2. 实验过程和核心代码（600 字以内，图文并茂）

写在前面：由于实验报告模版有字数限制，实验原理等更加详细的内容我会放在另一个 pdf 中，如有需要可以查看。

实验一： MPI+OpenMP 实现卷积操作

下面是在这个实验中的核心卷积函数：

```
void performConvolution(double* input, double* kernel, double* output,
                        int input_height, int input_width, int input_channel, int kernel_size,
                        int stride, int padding, int rank, int size, int output_channel) {
    int output_height = (input_height - kernel_size + 2 * padding) / stride + 1;
    int output_width = (input_width - kernel_size + 2 * padding) / stride + 1;

    int chunk_size = output_height / size;
    int start_row = rank * chunk_size;
    int end_row = (rank == size - 1) ? output_height : start_row + chunk_size;

    // Pre-calculate constant indices outside loops
    int kernel_size_sq = kernel_size * kernel_size;
    int input_hw = input_height * input_width;

    #pragma omp parallel for collapse(3)
    for (int oc = 0; oc < output_channel; oc++) {
        for (int oh = start_row; oh < end_row; oh++) {
            for (int ow = 0; ow < output_width; ow++) {
                double temp = 0.0;
                int output_index = oc * output_height * output_width + oh * output_width + ow;
                for (int ic = 0; ic < input_channel; ic++) {
                    for (int kh = 0; kh < kernel_size; kh++) {
                        for (int kw = 0; kw < kernel_size; kw++) {
                            int ih = oh * stride - padding + kh;
                            int iw = ow * stride - padding + kw;
                            if(ih >= 0 && ih < input_height && iw >= 0 && iw < input_width) {
                                int input_index = ic * input_hw + ih * input_width + iw;
                                int kernel_index = oc * input_channel * kernel_size_sq + ic * kernel_size_sq + kh * kernel_size + kw;
                                temp += input[input_index] * kernel[kernel_index];
                            }
                        }
                    }
                }
                output[output_index] = temp;
            }
        }
    }
}
```

- 首先，通过输入参数计算输出特征图的高度和宽度（output_height 和 output_width）。这里使用了常见的卷积操作公式来计算输出特征图的尺寸。
- 然后，根据当前进程的排名（rank）和总进程数（size），计算每个进程负责处理的输出特征图的行数。通过将输出特征图的高度（output_height）平均分配给每个进程，可以得到每个进程的起始行（start_row）和结束行（end_row）。
- 在并行计算的部分，使用 OpenMP 的并行 for 循环（#pragma omp parallel for）来并行化卷积操作的计算过程。通过 collapse(3)参数，将三个 for 循环进行合并并并行执行，提高计算效率。
- 在并行 for 循环中，首先迭代输出通道（oc），然后迭代每个进程负责处理的输出特征图的行（oh），最后迭代输出特征图的列（ow）。
- 在每个滑窗的位置，定义一个临时变量 temp 用于存储局部计算结果。计算输出特征图中当前位置的索引（output_index），通过公式 oc * output_height * output_width + oh * output_width + ow 计算得到。
- 在每个滑窗的位置，进行卷积计算。首先迭代输入通道（ic），然后迭代卷积核的高度（kh）和宽度（kw）。
- 根据滑窗的位置计算输入特征图中的对应位置（ih 和 iw），应用 padding 和 stride 的设置来映射到输入特征图中的位置。
- 检查输入特征图中的位置是否在合法范围内。如果是，计算输入特征图中的索引（input_index）和卷积核中的索引（kernel_index）。
- 使用临时变量 temp 累加输入特征图和卷积核对应位置的乘积。
- 将计算得到的局部结果存储到输出特征图的对应位置（output[output_index]）。

实验二，im2col 方法实现卷积操作

首先是对于 im2col 矩阵的构建：

```
int col_index = 0;
for (int oh = 0; oh < output_height; oh++) {
    for (int ow = 0; ow < output_width; ow++) {
        for (int ic = 0; ic < input_channel; ic++) {
            for (int kh = 0; kh < kernel_size; kh++) {
                for (int kw = 0; kw < kernel_size; kw++) {
                    int ih = oh * stride - padding + kh;
                    int iw = ow * stride - padding + kw;
                    if (ih >= 0 && ih < input_height && iw >= 0 && iw < input_width) {
                        int input_index = ic * input_height * input_width + ih * input_width + iw;
                        im2col[col_index] = input[input_index];
                    } else {
                        im2col[col_index] = 0;
                    }
                    col_index++;
                }
            }
        }
    }
}
```

- 通过嵌套的循环，迭代输出特征图的每个位置（oh 和 ow），输入通道（ic），以及卷积核的高度和宽度（kh 和 kw）。
- 根据滑窗的位置计算输入特征图中的对应位置（ih 和 iw），应用 padding 和 stride 的设置来映射到输入特征图中的位置。
- 检查输入特征图中的位置是否在合法范围内。如果是，在 im2col 矩阵中的当前列（由 col_index 确定）存储输入特征图中的对应位置的值（input[input_index]）。
- 如果输入特征图中的位置不在合法范围内（超出输入特征图的边界），在 im2col 矩阵中的当前列存储 0。
- 增加 col_index 的值，以便在下一次循环中存储下一个位置的值。

卷积函数：

```
void performConvolution(double* input, double* kernel, double* output, double* im2col,
                        int input_height, int input_width, int input_channel,
                        int kernel_size, int stride, int padding, int rank, int size, int output_channel) {
    int output_height = (input_height - kernel_size + 2 * padding) / stride + 1;
    int output_width = (input_width - kernel_size + 2 * padding) / stride + 1;

    // Calculate the range of output rows to be processed by this rank
    int chunk_size = output_height / size;
    int start_row = rank * chunk_size;
    int end_row = (rank == size - 1) ? output_height : start_row + chunk_size;

    // Calculate the number of elements in im2col matrix
    int im2col_height = output_height * output_width;
    int im2col_width = input_channel * kernel_size * kernel_size;

    // Perform matrix multiplication
    #pragma omp parallel for collapse(3)
    for (int oc = 0; oc < output_channel; oc++) {
        for (int oh = start_row; oh < end_row; oh++) {
            for (int ow = 0; ow < output_width; ow++) {
                double sum = 0;
                for (int col = 0; col < im2col_width; col++) {
                    int kernel_index = oc * im2col_width + col;
                    int im2col_index = (oh * output_width + ow) * im2col_width + col;
                    sum += kernel[kernel_index] * im2col[im2col_index];
                }
                int output_index = oc * output_height * output_width + oh * output_width + ow;
                output[output_index] = sum;
            }
        }
    }
}
```

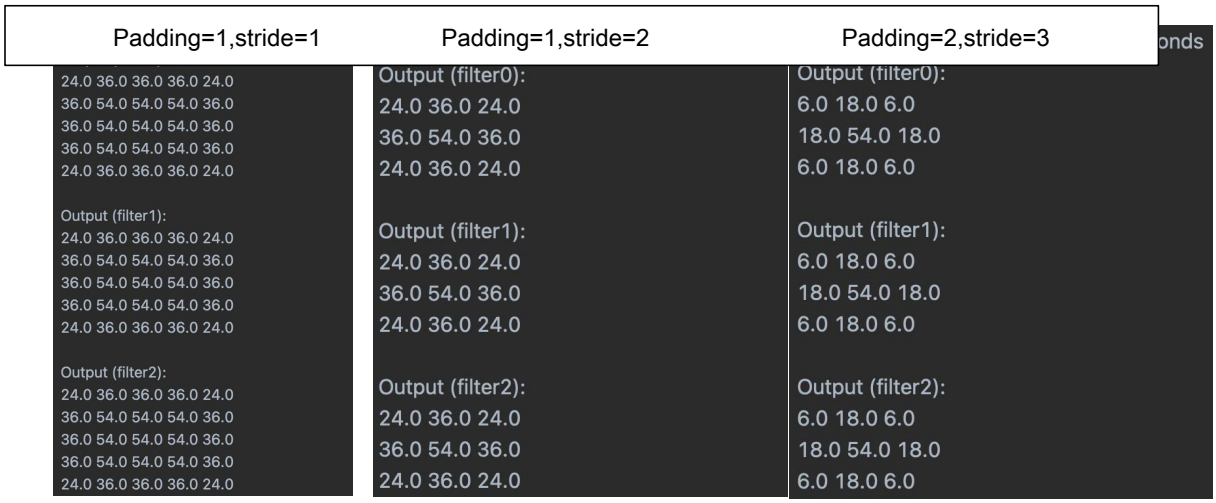
- 通过 OpenMP 的并行 for 循环（#pragma omp parallel for collapse(3)）将三个 for 循环进行合并并并行执行。这样可以并行处理输出通道（oc）、输出特征图的行（oh）和输出特征图的列（ow）。
- 在并行循环中，首先初始化局部变量 sum 为 0，用于存储每个输出特征图像素的计算结果。

- 对于每个输出像素，迭代 im2col 矩阵的列（col）进行矩阵乘法计算。每一列对应 im2col 中的一个滑窗区域，每个滑窗区域与卷积核进行元素级乘法，并将乘积结果累加到 sum 中。
- 通过索引计算 kernel 中的索引（kernel_index），用于获取卷积核中与当前滑窗区域对应的权重。
- 通过索引计算 im2col 中的索引（im2col_index），用于获取 im2col 矩阵中与当前滑窗区域对应的值。
- 将卷积操作的结果（sum）存储到输出特征图的对应位置。
- 通过索引计算输出特征图中的索引（output_index），用于确定输出特征图中的位置。

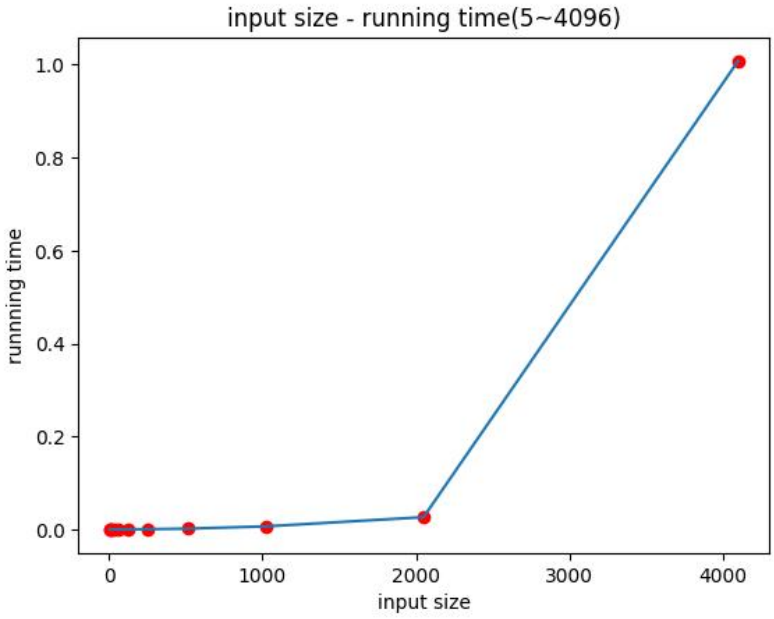
3. 实验结果（500 字以内，图文并茂）

实验一的结果:

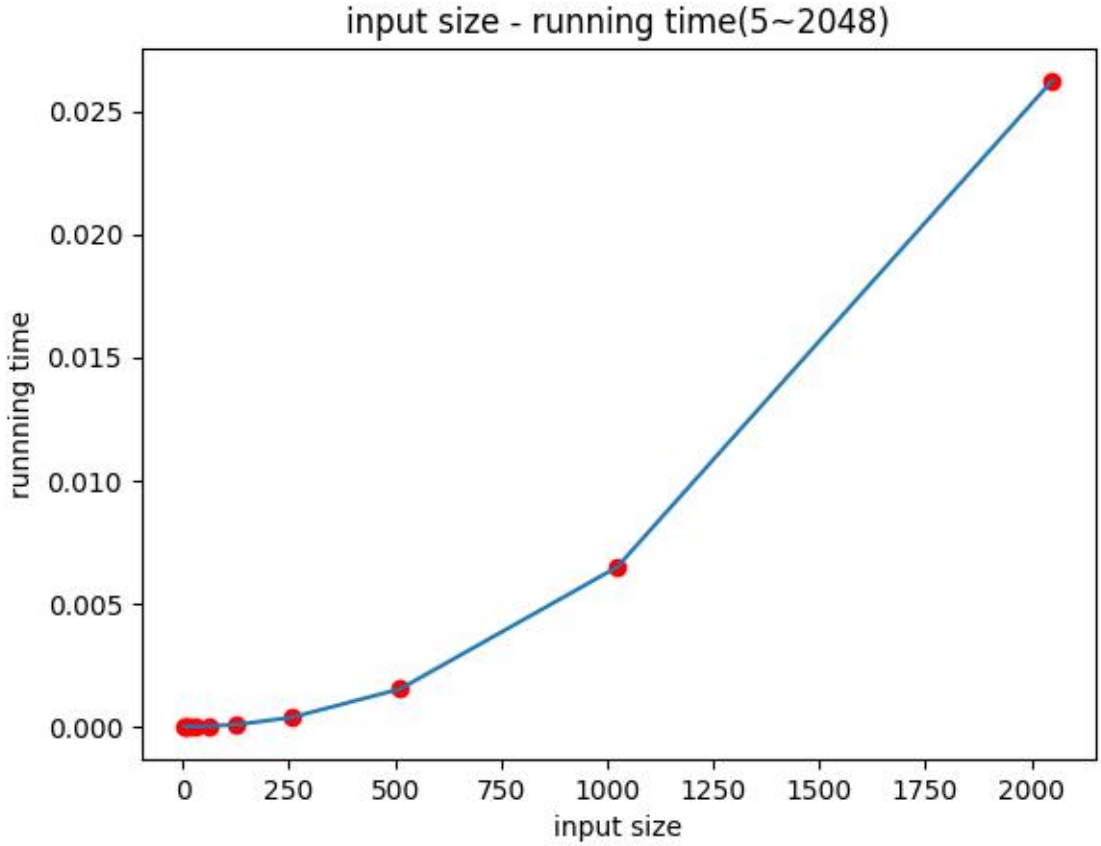
首先是三组不同步长的实验：输入矩阵为 5*5，初始化为 1，核为 3*3，初始化为 2.先用数据规模小的来判断正确性



Input size	5	8	16	32	64	128	256	512	1024	2048	4096
Running time	0	0	0	0	0.000018	0.000073	0.000363	0.001517	0.006479	0.026229	1.002974



由于2048到4096这个输入的时间跨度比较大，无法清晰看到前面的数据，所以单独画出来0～2048的折线图：



这张图片包含了从5～2048的输入和运行时间，通过分析可知，这条折线的斜率在不断增大，并且大致斜率为4，并且在数据规模变大的时候斜率超过了4。斜率为4是因为相邻两次实验的输入是在长和宽上都增加了一杯，这就导致在做卷积过程中的时候实际计算量要增加了四倍，所以运行时间会增加四倍，至于后面的斜率逐渐超过了4，可能的原因是这个时候运行时间的主体可能就不是卷积计算，而是其他的部分，比如数据读取和寻址之类的。

4096 输入的结果显示:

```
int main() {
    int rank, size;
    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int input_height = 4096;
    int input_width = 4096;
    int input_channel = 3;
    int kernel_size = 3;
    int stride = 1;
```

更改编辑框高度

标准输入

提交代码

运行结果

```
======== ERROR ========

======== OUTPUT ========
Elapsed Time (Rank 0): 1.002974 seconds
Output (filter0):
24.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0 36.0
```

实验二:im2col 实现卷积操作

首先是三组不同步长的实验：输入矩阵为 5*5，初始化为 1，核为 3*3，初始化为 2.先用数据规模小的来判断正确性

Padding=1,stride=1	Padding=1,stride=2	Padding=2,stride=3

```
Elapsed Time (Rank 0): 0.000000 seconds
Output (filter0):
24.0 36.0 36.0 36.0 24.0
36.0 54.0 54.0 54.0 36.0
36.0 54.0 54.0 54.0 36.0
36.0 54.0 54.0 54.0 36.0
24.0 36.0 36.0 36.0 24.0

Output (filter1):
24.0 36.0 36.0 36.0 24.0
36.0 54.0 54.0 54.0 36.0
36.0 54.0 54.0 54.0 36.0
36.0 54.0 54.0 54.0 36.0
24.0 36.0 36.0 36.0 24.0

Output (filter2):
24.0 36.0 36.0 36.0 24.0
36.0 54.0 54.0 54.0 36.0
36.0 54.0 54.0 54.0 36.0
36.0 54.0 54.0 54.0 36.0
24.0 36.0 36.0 36.0 24.0
```

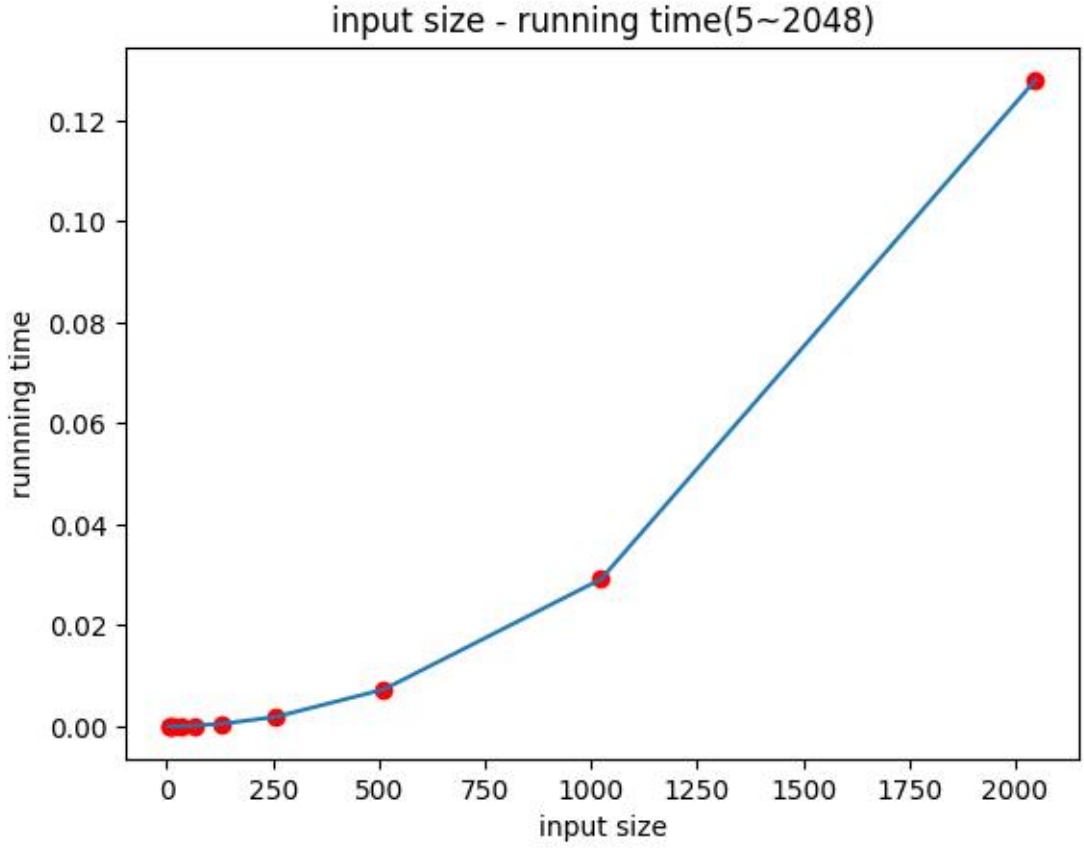
```
Elapsed Time (Rank 0): 0.000001 seconds
Output (filter0):
24.0 36.0 24.0
36.0 54.0 36.0
24.0 36.0 24.0

Output (filter1):
24.0 36.0 24.0
36.0 54.0 36.0
24.0 36.0 24.0

Output (filter2):
24.0 36.0 24.0
36.0 54.0 36.0
24.0 36.0 24.0
```

```
Elapsed Time (Rank 0): 0.000000 seconds
Output (filter0):
6.0 18.0 6.0
18.0 54.0 18.0
6.0 18.0 6.0

Output (filter1):
6.0 18.0 6.0
18.0 54.0 18.0
6.0 18.0 6.0

Output (filter2):
6.0 18.0 6.0
18.0 54.0 18.0
6.0 18.0 6.0
```

Input size	5	8	16	32	64	128	256	512	1024	2048

Running time	0	0	0	0	0.0000136	0.0000461	0.001831	0.007304	0.029143	0.127968

这是输入-时间折线图：



可以看到整体的变化趋势和滑窗法很像，都是斜率在逐渐增大，且大致为 4。

4. 实验感想（200 字以内）

本次试验主要是回顾了卷积的两种求法，一种病是滑窗法，另外一种是 im2col 方法。并且要将这两种结合 MPI 以及 OpenMP 这两种并行困来实现，其实这两种方法的原理并不难，难点在于为了实现更快的检索速度，我选择了用一维数组来表示所有需要存储的矩阵，这就导致我需要很精确的将一维索引和多位索引之间进行转换，因此我在编写代码的过程中经常出错，甚至还重新推导了好几次，确实是有点折磨人。从实验结果上看，并行程序的效率确实是远远大于普通编写的程序。

虽然实验要求的一些卷积核以及输入矩阵的一些参数都非常的规范且友好，但我还是将他们每一个都单独设置了变量，这样就可以按照要求随意更改超参数，并且效果也是不错的。通过这次实验我感受到一门技术的开创是难得的，这项技术的应用以及各领域优化更是一个大工程，尤其是在高性能计算领域，每一点点的提升经过成千上万的参数规模下都会获得巨大的进步。