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	**超算大作业**	专业（方向）	**计算机科学与技术**
学号	**21307265**	姓名	**童齐嘉**
Email	**3133261905@qq.com**	完成日期	**2023.06.17**

1. 实验目的（200 字以内）

为了实践检验本学期所学知识，掌握基本的并行程序的设计与实现，加深对卷积过程的理解，我按照要求，通过两种不同的方式实现了卷积操作，完成了本次大作业要求的两个实验。

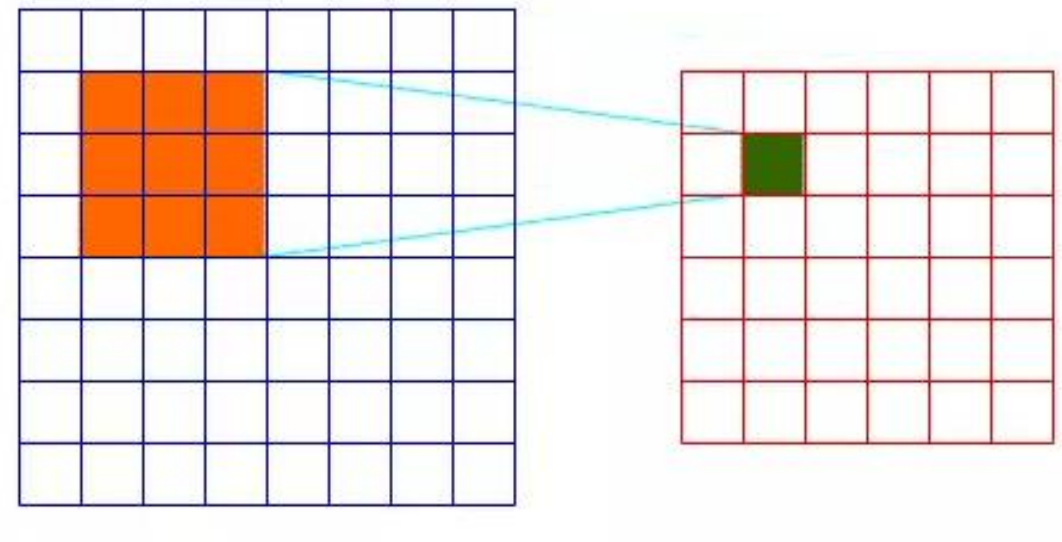
2. 实验过程和核心代码（600 字以内，图文并茂）

任务一：使用 MPI 和 OPENMP 实现卷积操作

首先，我们来描述一下滑窗法卷积操作：

- 取一个卷积核，将其放置在被卷积对象上按照步长移动
- 对其中的每一个重合区块，采取每格对应累乘，最后累加的形式，计算出输出矩阵中的一个单元。

对于实验要求所给的三维输入矩阵和三维输出矩阵，我们只需要嵌套六层循环即可解决主要计算步骤



对于任务一，我按照进程数将输入矩阵的行进行划分，然后分别计算出输出矩阵的结果，最终累加到根进程中：

```
int col = H;
int rows_per_proc = H / num_procs;
int start_row = rank * rows_per_proc;
int end_row = start_row + rows_per_proc - 1;
```

在计算好需要并行的行后，我们使用 OPENMP 划分了并行区域，实现了卷积的并行计算：

```
#pragma omp parallel for collapse(3)
    for (int i = 0; i < 3; i++)
        for(int j = start_row; j < end_row - S; j+=S)
            for(int k = 0; k < col - S + 1; k+=S)
                for(int ii = 0; ii < 3; ii++)
                    for(int jj = 0; jj < K; jj++)
                        for(int kk = 0; kk < K; kk++)
                            rec[j/S][k/S] += input[i][j + jj][k + kk] * ker[ii][jj][kk];
```

当然，最后需要将结果累加归结回根进程：

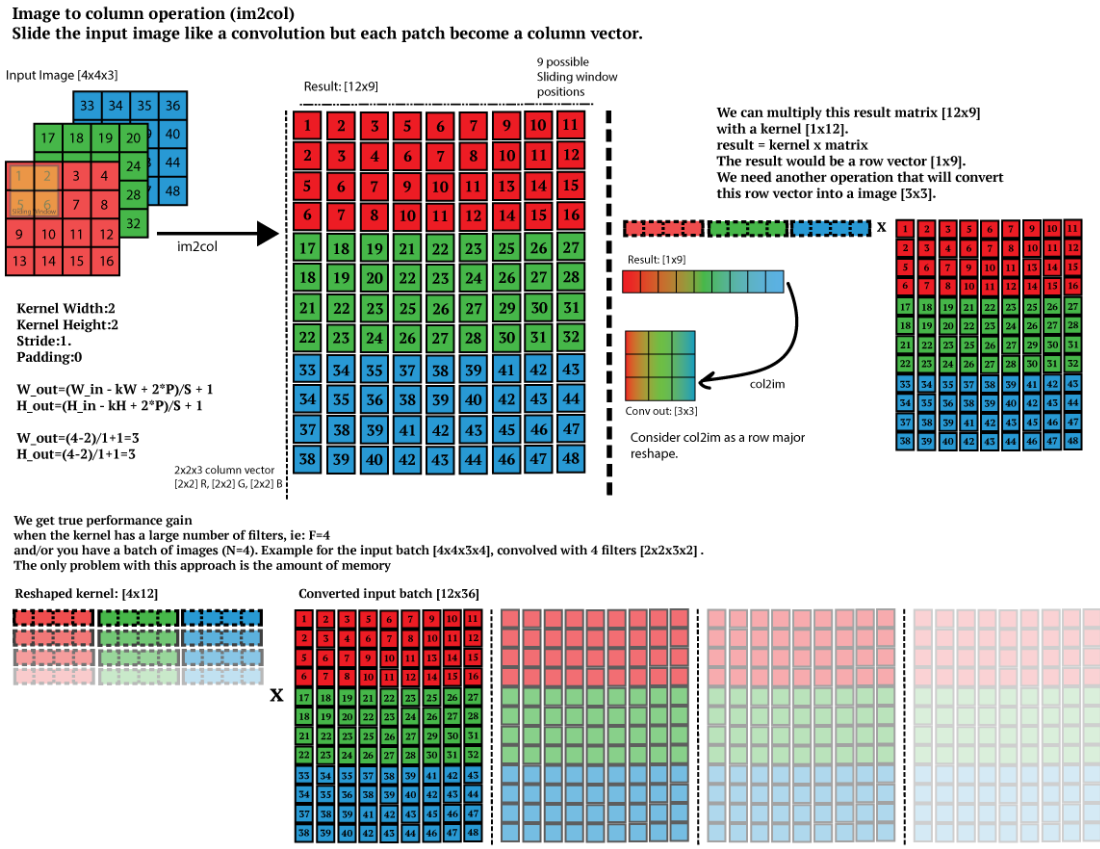
```
// 收集output
MPI_Reduce((void *)&rec, (void *)&output, W * W, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
```

至此，我们实现了所有的主体部分，再套上 MPI 框架，即可完成实验。

任务二：im2col 方法实现卷积

我们先来描述一下 im2col 卷积方法：

- 先按照步长，将矩阵划分为 N 个与卷积核大小相同的卷积区块。
- 然后将每个区块纵向展开为一个列向量，将所有的列向量拼接成一个 X 矩阵。
- 随后，将卷积核横向展开为一个行向量，按照形状差异复制 M 行卷积核并拼接成 Y 矩阵。
- 接着，使 XY 矩阵相乘，得到 Z 矩阵
- 最后，将 Z 矩阵还原回输出矩阵的形状



对于任务二，我按照 X 矩阵的行划分并行区域，分别计算出对于 output 矩阵的值：

```
int cols_per_proc = W*W / num_procs;
int start_col = rank * cols_per_proc;
int end_col = start_col + cols_per_proc - 1;
```

之后，需要使用 X 矩阵的列坐标倒推回 input 矩阵的对应单元的位置：

```
#pragma omp parallel for collapse(1)
    // 将input转换为X矩阵，X的大小为K*K*3行，W*W列。i:0-W*W
    for(int i = start_col; i < end_col; i++){ // 对于给定范围的一列
        int inp_row = (i / W) * S;  // 逆向求回其在输入矩阵的位置
        int inp_col = (i % W) * S;
        int row = 0;  // 在一列中的行位置
        for(int kk = 0; kk < 3; kk++)  // 走过input中，卷积核的层数
            for(int ii = inp_row; ii < K; ii++) // 走过input中，卷积核的行数
                for(int jj = inp_col; jj < K; jj++){  // 走过input中，卷积核对应的列数
                        X[row][i] = input[kk][ii][jj];
                        row ++;
                    }
    }
```

随后，求出 Z 矩阵并变换回 output 矩阵

```
#pragma omp parallel for collapse(2)
    for(int i = start_col; i < end_col; i++)
        for(int j = 0; j < 3*K*K; j++){
            Z[i] += X[j][i] * Y[0][j];  // 计算Z矩阵的值
        }

    // 将Z还原为输出特征output
#pragma omp parallel for collapse(1)
    for (int j = start_col; j <= end_col; j++) { // 遍历当前处理单元负责的每一个元素
        output[(j / W)][(j % W)] = Z[j]; // 将Z矩阵中的元素赋给output矩阵
    }
```

至此，我们实现了所有的主体部分，再套上 MPI 框架，即可完成实验。

3. 实验结果（500 字以内，图文并茂）

任务一：由于我们并未做输入端处理，所以所有的输入变动都选择直接在定义中更改完成：

```
# define H 512
# define K 3
# define S 1
# define W ((H-K)/S + 1)
```

实验结果统计表如下：

大小/步长	1	2	3
256	0.009684s	0.002208s	0.001905s
1024	0.165227s	0.037665s	0.017881s
4096	2.035725s	0.651469s	0.258217s

可以看到，即使规模到了较大的程度，用时依旧较短，说明了我们程序的并行化效果不错

部分截图展示：

```
summer@ubuntu:~/Documents/SCEW$ mpicc -fopenmp task1.c -o task1
summer@ubuntu:~/Documents/SCEW$ mpirun -n 16 ./task1
input size 3*N*N, N = 4096
stride = 3
Process 6 finished in 0.253273 seconds.
Process 1 finished in 0.255007 seconds.
Process 2 finished in 0.258217 seconds.
Process 3 finished in 0.255508 seconds.
Process 4 finished in 0.255283 seconds.
Process 5 finished in 0.256573 seconds.
Process 0 finished in 0.257712 seconds.
Process 8 finished in 0.254869 seconds.
Process 9 finished in 0.255247 seconds.
Process 10 finished in 0.257092 seconds.
Process 11 finished in 0.254717 seconds.
Process 12 finished in 0.255200 seconds.
Process 15 finished in 0.256770 seconds.
Process 7 finished in 0.259293 seconds.
Process 13 finished in 0.261547 seconds.
Process 14 finished in 0.265215 seconds.

finish
summer@ubuntu:~/Documents/SCEW$
```

```
summer@ubuntu:~/Documents/SCEW$ mpicc -fopenmp task1.c -o task1
summer@ubuntu:~/Documents/SCEW$ mpirun -n 16 ./task1
input size 3*N*N, N = 1024
stride = 2
Process 2 finished in 0.032011 seconds.
Process 1 finished in 0.033447 seconds.
Process 0 finished in 0.034653 seconds.
Process 4 finished in 0.032799 seconds.
Process 5 finished in 0.034662 seconds.
Process 6 finished in 0.033953 seconds.
Process 7 finished in 0.032664 seconds.
Process 8 finished in 0.032695 seconds.
Process 9 finished in 0.034529 seconds.
Process 11 finished in 0.032953 seconds.
Process 12 finished in 0.032377 seconds.
Process 13 finished in 0.033437 seconds.
Process 14 finished in 0.032825 seconds.
Process 15 finished in 0.034292 seconds.
Process 3 finished in 0.037665 seconds.
Process 10 finished in 0.037086 seconds.

finish
summer@ubuntu:~/Documents/SCEW$
```

任务二：

PS: 由于任务二我使用的方法并行化效果不算好，所以选择了数据规模较小的一组进行对比测试。

实验结果统计表如下：

大小/步长	1	2	3
32	0.001214s	0.000441s	0.000197s
128	0.003083	0.001150s	0.000508s
512	0.037734s	0.012289s	0.006862s

可以看到，即使规模较小时程序的运行时间基本符号预期，但事实上当规模很大时程序表现较差，这与我的代码的并行化程度较差有关。

部分截图展示：

```
summer@ubuntu:~/Documents/SCEW$ mpicc -fopenmp task2.c -o task2
summer@ubuntu:~/Documents/SCEW$ mpirun -n 16 ./task2
input size 3*N*N, N = 512
stride = 1
Process 1 finished in 0.037734 seconds.
Process 2 finished in 0.037576 seconds.
Process 3 finished in 0.037485 seconds.
Process 4 finished in 0.037462 seconds.
Process 5 finished in 0.037805 seconds.
Process 6 finished in 0.037499 seconds.
Process 7 finished in 0.037402 seconds.
Process 8 finished in 0.037514 seconds.
Process 9 finished in 0.037452 seconds.
Process 10 finished in 0.037418 seconds.
Process 11 finished in 0.037502 seconds.
Process 12 finished in 0.037410 seconds.
Process 13 finished in 0.037501 seconds.
Process 14 finished in 0.037413 seconds.
Process 15 finished in 0.037431 seconds.
Process 0 finished in 0.037655 seconds.
finish
summer@ubuntu:~/Documents/SCEW$
```

```
summer@ubuntu:~/Documents/SCEW$ mpicc -fopenmp task2.c -o task2
summer@ubuntu:~/Documents/SCEW$ mpirun -n 16 ./task2
input size 3*N*N, N = 128
stride = 2
Process 1 finished in 0.000552 seconds.
Process 2 finished in 0.000547 seconds.
Process 3 finished in 0.000556 seconds.
Process 4 finished in 0.000524 seconds.
Process 5 finished in 0.000519 seconds.
Process 6 finished in 0.000512 seconds.
Process 7 finished in 0.000472 seconds.
Process 8 finished in 0.000462 seconds.
Process 9 finished in 0.000483 seconds.
Process 10 finished in 0.000473 seconds.
Process 11 finished in 0.000459 seconds.
Process 12 finished in 0.000446 seconds.
Process 13 finished in 0.000457 seconds.
Process 14 finished in 0.000451 seconds.
Process 15 finished in 0.000438 seconds.
Process 0 finished in 0.000601 seconds.
finish
summer@ubuntu:~/Documents/SCEW$
```

4. 实验感想（200 字以内）

感想：如何较为效率地编写一份并行程序?

答：从本次实验的经验来看，面对较为复杂的程序时，首先应该写好一份串行程序，先完成要求，确保编译运行，结果验证通过后，再寻找一个贯穿始终，可以并行计算的变量，将其划分，把串行程序改为并行程序。这样正确率和效率都会较好，可以有效避免无从下手的问题。