**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 | **超级计算原理与实践大作业** | 专业（方向） | **计算机科学与技术（系统结构）** |
| 学号 | **21307358** | 姓名 | **曾慧蕾** |
| Email | **znghlei@mail2.sysu.edu.cn** | 完成日期 | **2023/6/7** |

# 实验目的（200字以内）

* **任务一：**

MPI+OpenMP实现直接卷积（滑窗法），输入从256增加至4096或者输入从32增加至512。要考虑通过填充（padding）配合步幅完成CNN操作。

* **任务二：**

用im2col方法结合任务一实现的GEMM实现卷积操作。输入从256增加至4096或者输入从32增加至512。

* 两个实验都要求实现2D, height\*width，通道channel(depth)设置为3，Kernel (Filter)大小设置为3\*3\*3，个数为3。不需要考虑bias(b)，bias设置为0，步幅(stride)分别设置为1，2，3。
* 输入：input和kernel（filter）；输出：卷积结果和时间。

# 实验过程和核心代码

在两个实验中，我选择输入从32增加到512，（32，128，256）调整在最开头的HEIGHT与WIDTH处，并基于步幅stride分别为1、2、3的调整，每个任务共计实验9次。下面是两个实验的细节。

* 任务一：

1. 首先初始化输入图像和卷积核。输入图像是一个 HEIGHTxWIDTHxDEPTH 的三维数组，表示一个 HEIGHTxWIDTH 的彩色图像。卷积核是一个 3x3x3x3 的四维数组，表示三个 3x3 的卷积核，每个卷积核有三个通道。

啊啊啊if (rank == 0) {

        for (i = 0; i < HEIGHT; i++) {

            for (j = 0; j < WIDTH; j++) {

                for (k = 0; k < DEPTH; k++) {

                    n = rand() % 10;

                        if (n >= 0 && n < 4) input[i][j][k] = 1;

                        else if (n >= 4 && n < 7) input[i][j][k] = -1;

                        else input[i][j][k] = 0;

                }

            }

        }

        for (n = 0; n < FILTER\_NUM; n++) {

            for (p = 0; p < FILTER\_SIZE; p++) {

                for (q = 0; q < FILTER\_SIZE; q++) {

                    for (k = 0; k < DEPTH; k++){

                        i = rand() % 10;

                        if (i >= 0 && i < 4) filter[n][p][q][k] = 1;

                        else if (i >= 4 && i < 7) filter[n][p][q][k] = -1;

                        else filter[n][p][q][k] = 0;

                    }

                }

            }

        }

    }

这一部分随机对每个元素赋-1、0、1的值，也方便后续计算。

在初始化完成后，向其他进程发送数据。

 // 把filter发送给多个进程

    MPI\_Bcast(filter, FILTER\_NUM \* FILTER\_SIZE \* FILTER\_SIZE \* DEPTH, MPI\_INT, 0, MPI\_COMM\_WORLD);

    MPI\_Bcast(input, HEIGHT \* WIDTH \* DEPTH, MPI\_INT, 0, MPI\_COMM\_WORLD);

1. 对输入图像进行填充。填充的目的是为了保持卷积后图像的大小不变。填充的大小根据不同的步长有不同的值。最终的目的都是通过填充，使最终卷积出来的图像大小与原图像相同。

公式推导：[(height+2\*pid\_size -3)/stride]+1=h,

推出：pad\_size=[(height-1)\*s+3-h]/2

int pad\_size[3];

    for(i = 0; i < 3; i++){ //根据不同的步长有不同的填充值

        pad\_size[i] = ((HEIGHT - 1) \* stride[i] + 3 - HEIGHT) / 2;

    }

    m = 0; //当要修改stride的值时，改这里

int input\_pad[HEIGHT + 2 \* pad\_size[m]][WIDTH + 2 \* pad\_size[m]][DEPTH];

padding(input,  pad\_size[m], input\_pad);

padding函数使用了三层循环来遍历输入。其中，第一层循环遍历行，第二层循环遍历列，第三层循环遍历通道。如果当前像素在输入图像的边缘上，则将其赋值为 0；否则，计算出对应的输入图像中的像素位置，并将其赋值给输出图像中的像素。

void padding(int input[HEIGHT][WIDTH][DEPTH],  int pad\_size, int output[HEIGHT + 2 \* pad\_size][WIDTH + 2 \* pad\_size][DEPTH]) { //填充0

    int i, j, k;

    for (i = 0; i < HEIGHT + 2 \* pad\_size; i++){

        for (j = 0; j < WIDTH + 2 \* pad\_size; j++){

            for (k = 0; k < DEPTH; k++){

                if (i < pad\_size || i >= HEIGHT + pad\_size || j < pad\_size || j >= WIDTH + pad\_size){

                    output[i][j][k]=0;

                }

                else{

                    int row = i - pad\_size;

                    int col = j - pad\_size;

                    output[i][j][k] = input[row][col][k];

                }

            }

        }

    }

}

1. 进行卷积操作。卷积操作是将卷积核在输入图像上滑动，计算每个位置的卷积结果。这个过程可以使用 OpenMP 将输入图像和卷积核分别在不同的进程上进行计算，然后将计算结果汇总到主进程中。

#pragma omp parallel for private(j, k, l, m, n, p, q) shared(input, filter, output, stride, pad\_size) schedule(static)

    for (i = 0; i < end-start; i++){

        for (n = 0; n < FILTER\_NUM; n++){

       // for (i = 0; i < HEIGHT; i++){

            for (j = 0; j < WIDTH; j++){

                for (k = 0; k < DEPTH; k++){

                    for (p = 0; p < FILTER\_SIZE; p++){

                        for (q = 0; q < FILTER\_SIZE; q++){

                            m = i \* stride + p;

                            l = j \* stride + q;

                            if (m >= 0 && m < HEIGHT + 2 \* pad\_size && l >= 0 && l < WIDTH + 2 \* pad\_size){

                                part\_output[i][j][n] += filter[n][p][q][k] \* input[m][l][k];

                            }

                        }

                    }

                }

            }

        }

    }

1. 汇总卷积结果。由于卷积操作是在多个进程上进行的，因此需要将每个进程的卷积结果汇总到主进程中。这一步直接在convolution函数中完成。

    // 在主进程中收集所有进程的运行结果

    MPI\_Gather(part\_output, (end-start) \* WIDTH \* FILTER\_NUM, MPI\_INT, output, (end-start) \* WIDTH \* FILTER\_NUM, MPI\_INT, 0, MPI\_COMM\_WORLD);

}

1. 输出卷积结果和计算时间。只用判断是不是主进程就行。

    if (rank == 0) {

        printf("Convolution result:\n");

        for (n = 0; n < FILTER\_NUM; n++) {

            printf("Filter %d:\n", n);

            for (i = 0; i < HEIGHT; i++) {

                for (j = 0; j < WIDTH; j++) {

                    printf("%3d ", output[i][j][n]);

                }

                printf("\n");

            }

            printf("\n");

        }

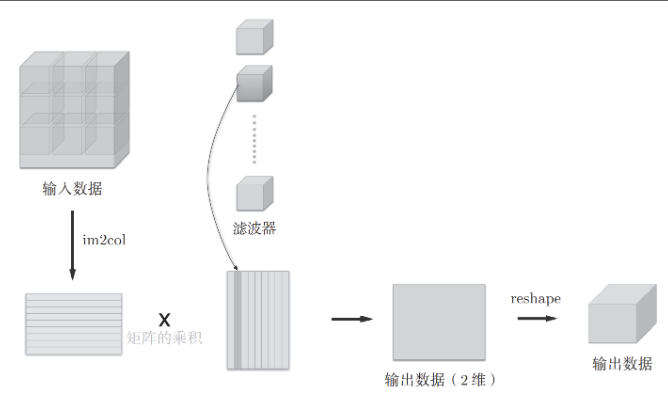
        printf("Elapsed time = %f seconds\n", t\_end-t\_start);

    }

* 任务二：

任务二与任务一的实验目的是一样的，因此实验思路也类似，只是在卷积前对输入图像的操作有所不同。任务二在卷积前不再对图像进行padding操作，而是使用im2col方法，将输入图像展开为二维矩阵。展开后的矩阵的每一列对应于输入图像中的一个滤波器的卷积窗口。

1. 首先初始化输入图像和卷积核。输入图像是一个 HEIGHTxWIDTHxDEPTH 的三维数组，表示一个 HEIGHTxWIDTH 的彩色图像。卷积核是一个 3x3x3x3 的四维数组，表示三个 3x3 的卷积核，每个卷积核有三个通道。实现思路与任务一相同。
2. 进行卷积操作。在卷积前，先使用im2col方法对输入图像进行调整。以下是im2col方法的原理：



Im2col方法使用六层循环来遍历输入图像和滤波器。在遍历到的每个位置上，我们计算出对应的滤波器的卷积窗口，并将窗口中的像素展开为一列，然后将所有列组合成一个二维矩阵。在展开过程中，我们还需要对输入图像进行 padding，以保证卷积后输出图像的大小与输入图像相同。

void im2col(int input[HEIGHT][WIDTH][DEPTH], int col[FILTER\_SIZE \* FILTER\_SIZE \* DEPTH][HEIGHT \* WIDTH], int stride) {

    int i, j, k, p, q, r;

    int index = 0;

    for (k = 0; k < DEPTH; k++) {

        for (p = 0; p < FILTER\_SIZE; p++) {

            for (q = 0; q < FILTER\_SIZE; q++) {

                for (i = 0; i < HEIGHT; i++) {

                    for (j = 0; j < WIDTH; j++) {

// int m = i + p - FILTER\_SIZE / 2;

// int n = j + q - FILTER\_SIZE / 2;

int m = i \* stride;

int n = j \* stride;

if (m >= 0 && m <= HEIGHT && n >= 0 && n <= WIDTH) {

                            col[index][i \* WIDTH + j] = input[m][n][k];

                        } else {

                            col[index][i \* WIDTH + j] = 0;//同时完成padding

                        }

                    }

                }

                index++; //针对每个滤波器展开input

            }

        }

    }

}

卷积操作是将卷积核在输入图像上滑动，计算每个位置的卷积结果。这个过程可以使用 OpenMP 进行并行计算，加快计算速度。实现思路与任务一相同。

1. 汇总卷积结果。由于卷积操作是在多个进程上进行的，因此需要将每个进程的卷积结果汇总到主进程中。实现思路与任务一相同。
2. 输出卷积结果和计算时间。实现思路与任务一相同。

# 实验结果

由于输出内容较多，仅截图将占很大篇幅来展示实验结果，比如下图所示，当图像面积很大的时候会很不方便展示结果：

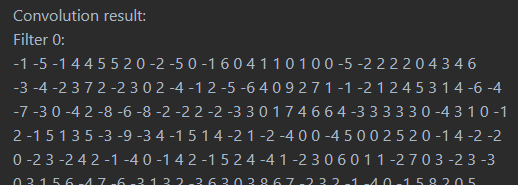


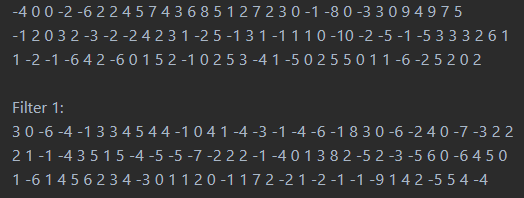
因此这里在第一次实验时截取每个filter有效输出的开头一段与结尾一段进行展示，随后仅展示filter0的结果（因为三个filter执行操作一样，输出结果也类似）。并在最后总结出表格

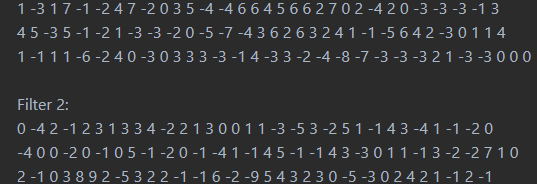
* 任务一：

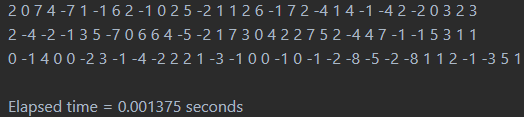
1. 当stride=1时：

HEIGHT=32, WIDTH=32:





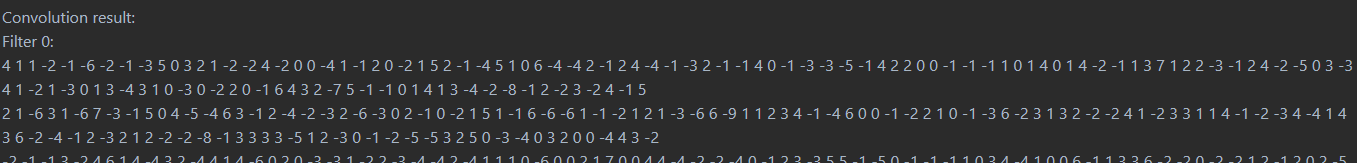


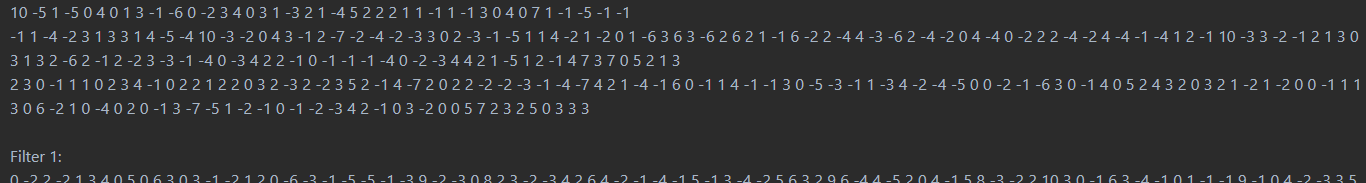
（n=1）

(n=2)

(n=4)

HEIGHT=128, WIDTH=128（局部）:



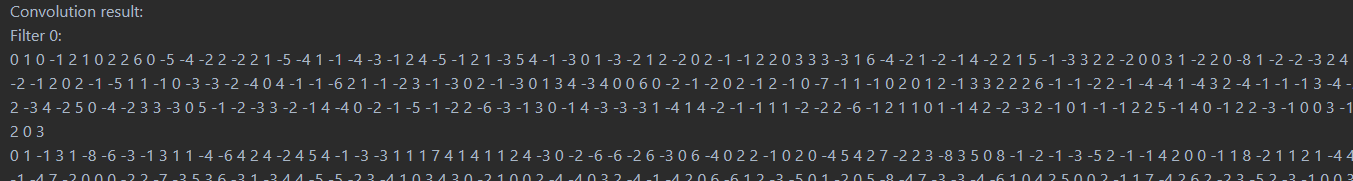


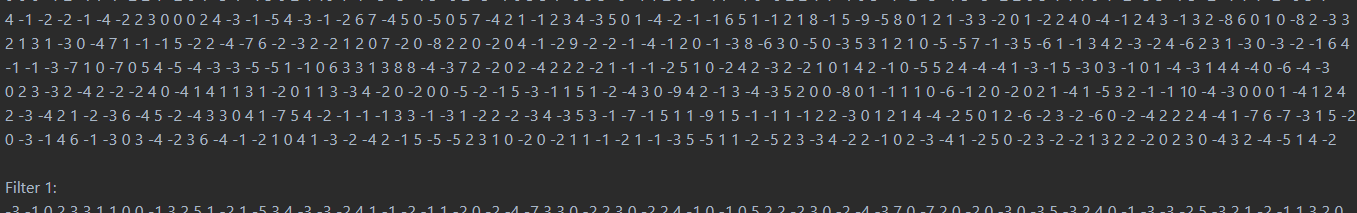
(n=1)

(n=2)

(n=4)

HEIGHT=256, WIDTH=256（局部）:





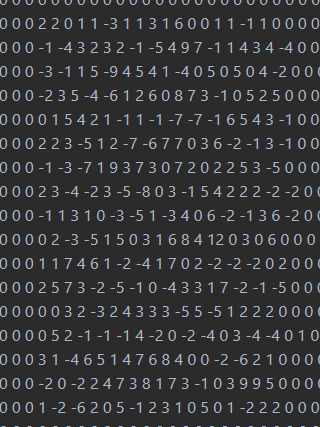
(n=1)

(n=2)

(n=4)

1. 当stride=2时：此时，由于padding的操作，会使得最后输出图像四周皆为0，而中间有效部分为真实的卷积大小。

HEIGHT=32, WIDTH=32:

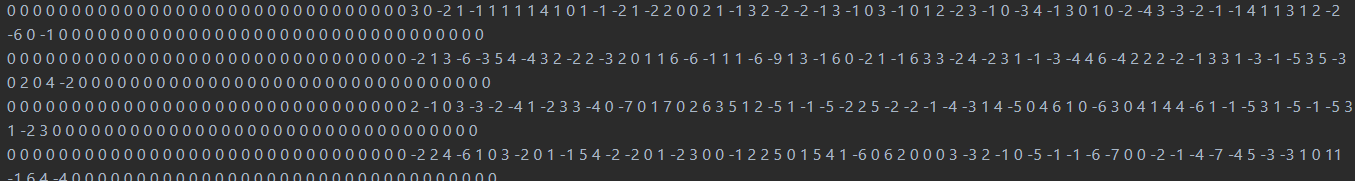
 ←Filter 0

(n=1)

(n=2)

(n=4)

HEIGHT=128, WIDTH=128（局部）:

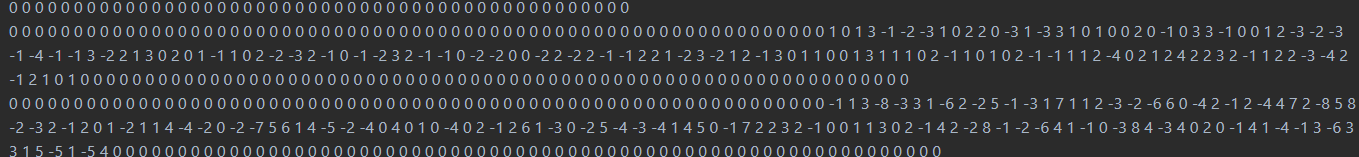


(n=1)

(n=2)

(n=4)

HEIGHT=256, WIDTH=256（局部）:



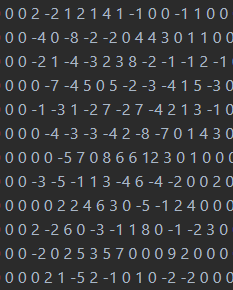
(n=1)

(n=2)

(n=4)

1. 当stride=3时：有效图像更小了！

HEIGHT=32, WIDTH=32:

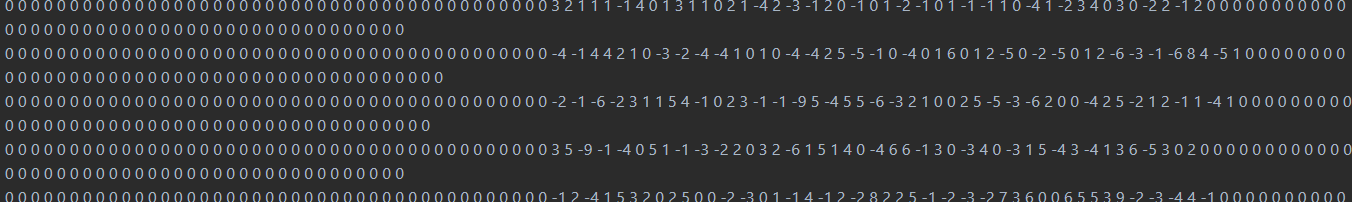
←Filter 0

（n=1）

（n=2）

（n=4）

HEIGHT=128, WIDTH=128（局部）:

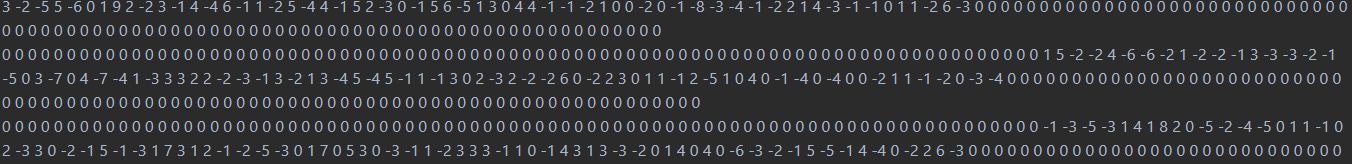


（n=1）

（n=2）

（n=4）

HEIGHT=256, WIDTH=256（局部）:



（n=1）

（n=2）

（n=4）

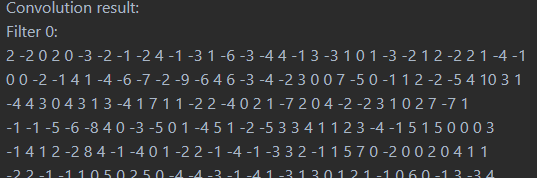
综上，可列表格(单位为毫秒，顺序按核数为：1/2/3展示)：

|  |  |  |  |
| --- | --- | --- | --- |
| 面积/步幅 | 1 | 2 | 3 |
| 32\*32 | 1375/769/523 | 1418/819/594 | 1559/963/742 |
| 128\*128 | 21544/11495/10315 | 22973/12780 | 24613/14807/31669 |
| 256\*256 | 85749/37934/21025 | 75055/41701/26081 | 102647/47477/31669 |

* 任务二

1. 当stride=1时：

HEIGHT=32, WIDTH=32:

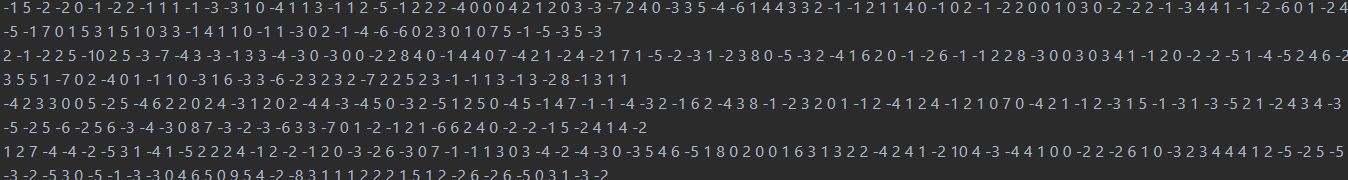


（n=1）

（n=2）

（n=4）

HEIGHT=128, WIDTH=128:

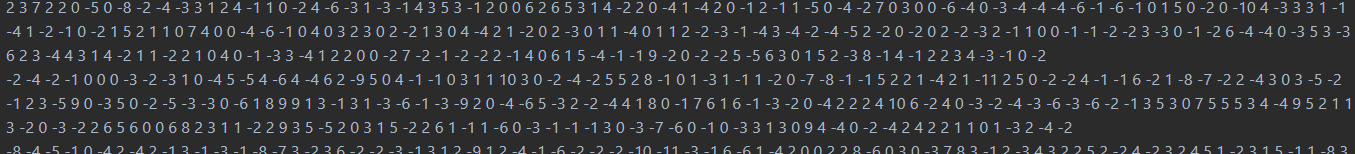


（n=1）

（n=2）

（n=4）

HEIGHT=256, WIDTH=256:



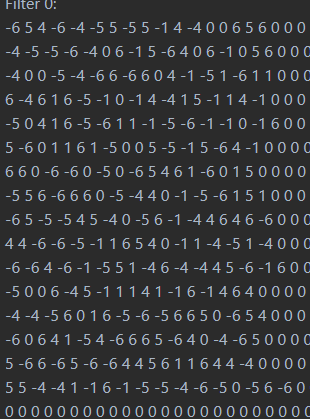
（n=1）

（n=2）

（n=4）

1. 当stride=2时：

HEIGHT=32, WIDTH=32:

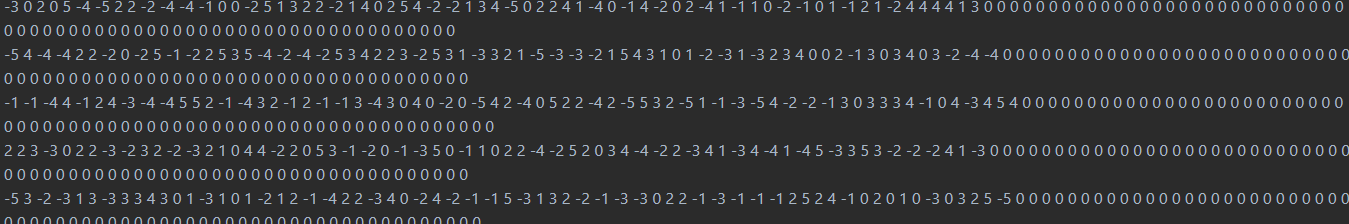


（n=1）

（n=2）

（n=4）

HEIGHT=128, WIDTH=128:

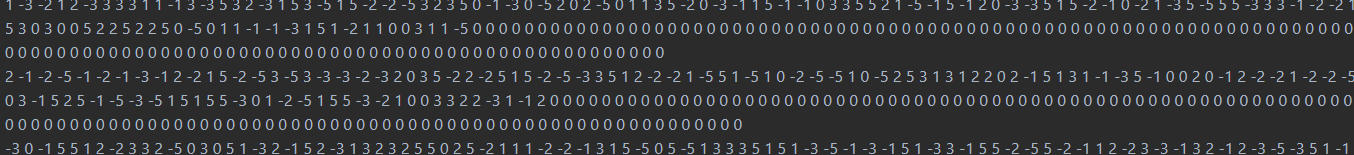


（n=1）

（n=2）

（n=4）

HEIGHT=256, WIDTH=256:



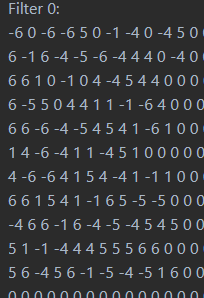
（n=1）

（n=2）

（n=4）

1. 当stride=3时：

HEIGHT=32, WIDTH=32:

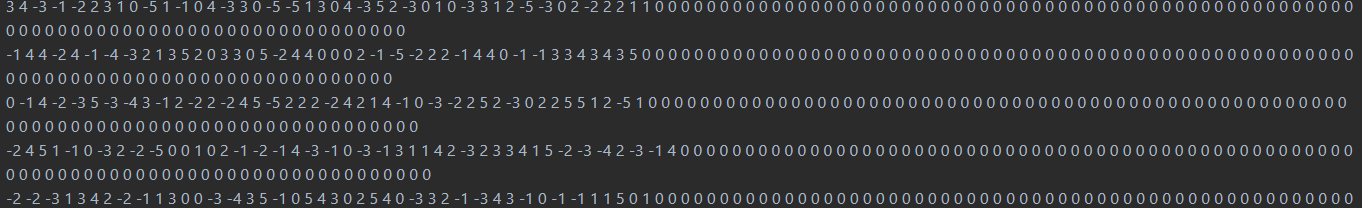


（n=1）

（n=2）

（n=4）

HEIGHT=128, WIDTH=128:

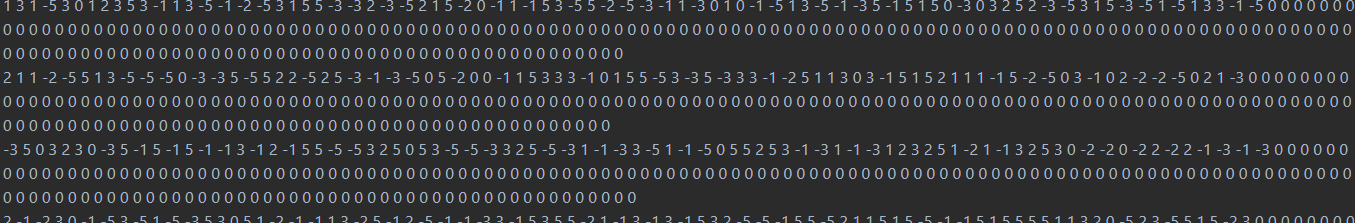


（n=1）

（n=2）

（n=4）

HEIGHT=256, WIDTH=256:



（n=1）

（n=2）

（n=4）

综上，可列表格(单位为毫秒，顺序按核数为：1/2/3展示)：

|  |  |  |  |
| --- | --- | --- | --- |
| 面积/步幅 | 1 | 2 | 3 |
| 32\*32 | 1302/858/667 | 1292/786/677 | 1564/778/600 |
| 128\*128 | 22548/13808/10281 | 26434/12740/8809 | 22164/13102/8784 |
| 256\*256 | 70567/43269/35212 | 70093/40495/27411 | 80598/39423/26760 |

# 实验感想

并行计算可提高计算速度，尤其是处理大规模数据时，可充分利用多核CPU的计算资源。本实验当核数增加时，计算时间可成比例的下降，如1核提升到2核的加速比通常都在45%~50%间，直接节省一半计算时间。

但是，并行计算也带来一些挑战。在这次实验中，需要考虑进程和线程之间的通信和同步问题。我出现的最大的问题就是通信问题：MPI\_Gather不知道该放哪个位置，一开始是直接放在主函数中，但是这样的做法在stride增长以后输出非常奇怪，在跟同学请教以后才知道可以放在convolution函数中，这才使输出正常。