中山大学计算机院本科生实验报告

（2024 学年秋季学期）

课程名称：高性能计算程序设计　　批改人：

实验	基于 CUDA 编程	专业（方向）	信息与计算科学
学号	22336044	姓名	陈圳煌
Email	2576926165@qq.com	完成日期	12.30

1. 实验目的

学习如何基于 CUDA 编程，并通过实现通用矩阵乘法和两种方式的卷积来熟悉 CUDA 编程，利用 GPU 的结构实现程序的并行化。学习 cuBLAS 和 cuDNN 这两个基于 CUDA 的库的一些基本用法，并与自己实现的方法进行性能上的比较，从而更好地掌握 CUDA 编程。

2. 实验过程和核心代码

task1：参考文件 task1.cu、task1_1.cu

```
// 随机初始化
void random_ints(float* a, int n)
{
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> dis(0.0, 5.0);

    for (int i = 0; i < n; i++)
    {
        a[i] = dis(gen);
    }
}
// 矩阵乘法
__global__ void mat_mul(float *A, float *B, float *C, int M, int N, int K)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    // 每个线程计算一个C[i][j]
    if (row < M && col < K)
    {
        float sum = 0;
        for (int i = 0; i < N; i++)
        {
            sum += A[row * N + i] * B[i * K + col];
        }
        C[row * K + col] = sum;
    }
}

int main()
{
    // 查看设备属性
    int device;
    cudaDeviceProp prop;
    cudaGetDevice(&device);
    cudaGetDeviceProperties(&prop, device);

    cout << "Number of thread blocks: " << prop.maxGridSize[0] << endl;
    cout << "Max threads per block: " << prop.maxThreadsPerBlock << endl;
```

```
    int M, N, K;
    cin >> M >> N >> K;
    float *A, *B, *C;
    float *d_A, *d_B, *d_C;

    int size_a = M * N * sizeof(float);
    int size_b = N * K * sizeof(float);
    int size_c = M * K * sizeof(float);

    // 分配设备内存
    cudaMalloc((void**)&d_A, size_a);
    cudaMalloc((void**)&d_B, size_b);
    cudaMalloc((void**)&d_C, size_c);

    // 分配主机内存以及随机初始化
    A = (float*)malloc(size_a);
    random_ints(A, M * N);
    B = (float*)malloc(size_b);
    random_ints(B, N * K);
    C = (float*)malloc(size_c);

    // 将数据从主机拷贝到设备
    cudaMemcpy(d_A, A, size_a, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, size_b, cudaMemcpyHostToDevice);

    // 设置线程块和网格大小
    int block_y = BLOCK_SIZE;
    int block_x = 512 / BLOCK_SIZE;
    dim3 dimBlock(block_x, block_y);
    dim3 dimGrid((K + block_x - 1) / block_x, (M + block_y - 1) / block_y);

    // 记录开始时间
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start, 0);

    // 进行矩阵乘法
    mat_mul<<<dimGrid, dimBlock>>>(d_A, d_B, d_C, M, N, K);

    // 记录结束时间
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);

    float time;
    cudaEventElapsedTime(&time, start, stop);

    // 销毁事件
    cudaEventDestroy(start);
    cudaEventDestroy(stop);

    // 将数据传回主机
    cudaMemcpy(C, d_C, size_c, cudaMemcpyDeviceToHost);
    cout << "Part Result: " << C[2233]  << "  " << C[6044] << "  " << C[M * K - 1] << endl;
    cout << "Used time: " << time << "ms" << endl;

    // 释放内存
    free(A);
    free(B);
    free(C);
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
    return 0;
}
```

首先为主机和 GPU 设备分配好数据的空间，在主机上随机初始化 A，B 后传输到 GPU 上，设置 Grid 的大小以及线程块的大小，根据分配的 CUDA 线程的编号分配对应的结果矩阵上的点的计算，最后将结果矩阵回传到主机上。

编译指令为： nvcc task1.cu -o task1

使用共享内存进行优化：

```
// 优化后的矩阵乘法
__global__ void mat_mul(float *A, float *B, float *C, int M, int N, int K)
```

```
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    // 用于存储当前线程块所需要的矩阵块
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    for (int t = 0; t < (N + BLOCK_SIZE - 1) / BLOCK_SIZE; t++)
    {
        if (row < M && t * BLOCK_SIZE + threadIdx.x < N)
        {
            As[threadIdx.y][threadIdx.x] = A[row * N + t * BLOCK_SIZE + threadIdx.x];
        }
        else
        {
            As[threadIdx.y][threadIdx.x] = 0;
        }

        if (col < K && t * BLOCK_SIZE + threadIdx.y < N)
        {
            Bs[threadIdx.y][threadIdx.x] = B[(t * BLOCK_SIZE + threadIdx.y) * K + col];
        }
        else
        {
            Bs[threadIdx.y][threadIdx.x] = 0;
        }
        // 用于同步
        __syncthreads();

        float sum = 0;
        for (int i = 0; i < BLOCK_SIZE; i++)
        {
            sum += As[threadIdx.y][i] * Bs[i][threadIdx.x];
        }

        __syncthreads();

        if (row < M && col < K)
        {
            C[row * K + col] += sum;
        }
    }
}
```

分配一个32×32的共享数组将矩阵A和B存储当前位置的邻近元素，类似于使用Cache来减少访存次数的原理，可以一定程度上提升运行速度。

task2：参考文件task2.cu

```
int main()
{
    int M, N, K;
    cin >> M >> N >> K;
    float *A, *B, *C;
    float *d_A, *d_B, *d_C;

    int size_a = M * N * sizeof(float);
    int size_b = N * K * sizeof(float);
    int size_c = M * K * sizeof(float);
    A = (float *)malloc(size_a);
    random_ints(A, M * N);
    B = (float *)malloc(size_b);
    random_ints(B, N * K);
    C = (float *)malloc(size_c);

    // 分配设备内存
    cudaMalloc((void**)&d_A, size_a);
    cudaMalloc((void**)&d_B, size_b);
    cudaMalloc((void**)&d_C, size_c);

    // 将数据从主机内存复制到设备内存
    cudaMemcpy(d_A, A, size_a, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, B, size_b, cudaMemcpyHostToDevice);
    // 创建cublas 句柄
    cublasHandle_t handle;
    cublasCreate(&handle);

    // 记录开始时间
```

```
    cudaEvent_t start, stop;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start, 0);

    const float alpha = 1.0f;
    const float beta = 0.0f;
    cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, M, K, N,
                &alpha, d_A, M, d_B, N, &beta, d_C, M);
    // 记录结束时间
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    float time;
    cudaEventElapsedTime(&time, start, stop);
    // 销毁事件
    cudaEventDestroy(start);
    cudaEventDestroy(stop);
    // 将数据从设备内存复制到主机内存
    cudaMemcpy(C, d_C, size_c, cudaMemcpyDeviceToHost);
    cout << "Part Result: " << C[2233]  << "  " << C[6044] << "  " << C[M * K - 1] << endl;
    cout << "Used time: " << time << "ms" << endl;
    cublasDestroy(handle);
    free(A);
    free(B);
    free(C);
    return 0;
}
```

与 task1 的过程类似，需要创建 cublas 句柄，在实现矩阵乘法部分调用 cublasSgemm 函数实现，最后要销毁句柄。

编译指令：nvcc task2.cu -o task2 -lcublas

task3：参考文件 task3.cu

```
// 卷积核
__constant__ float filter[9]=
{
    1.5, -2.5, 1.5,
    -2.5, 4.0, -2.5,
    1.5, -2.5, 1.5
};
// 初始化图像矩阵，并加上填充
void random_ints(float* a, int n, int d)
{
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, 255);

    for (int c = 0; c < 3; c++)
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                a[c * n * n + i * n + j] = 0.0;
            }
        }
    }

    for (int c = 0; c < 3; c++)
    {
        for (int i = 0; i < n - d; i++)
        {
            for (int j = 0; j < n - d; j++)
            {
                a[c * n * n + i * n + j] = 1.1 * dis(gen);
            }
        }
    }
}
// 卷积函数
__global__ void conv2d(float *mat, float *res, int n, int m, int padding, int stride)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < m && col < m)
    {
        float sum = 0.0;
        for (int kr = 0; kr <= 2; kr++)
```

```
        {
            for (int kc = 0; kc <= 2; kc++)
            {
                int i = row * stride + kr;
                int j = col * stride + kc;
                if (i >= 0 && i < n && j >= 0 && j < n)
                {
                    sum += mat[i * n + j] * filter[kr * 3 + kc];
                    sum += mat[n * n + i * n + j] * filter[kr * 3 + kc];
                    sum += mat[2 * n * n + i * n + j] * filter[kr * 3 + kc];
                }
            }
        }
        res[row * m + col] = sum;
    }
}
```

首先初始化图像矩阵，这边用 1.1×[0,255]之间的数来形成随机浮点数，并且手动计算结果矩阵的维度，将数据传入设备后，为结果矩阵的每个节点分配一个 CUDA 线程进行卷积计算。

编译指令：nvcc task3.cu -o task3

task4：参考文件 task4.cu

```
// 卷积运算
__global__ void conv_im2col(float *mat, float *res, int m)
{
    int s_block = 9 * m * m;
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;

    if (row < m && col < m)
    {
        float sum = 0.0;
        for(int i = 0; i < 9; i++)
        {
            sum += filter[i] * mat[row * (9 * m) + i * m + col];
            sum += filter[i] * mat[s_block + row * (9 * m) + i * m + col];
            sum += filter[i] * mat[2 * s_block + row * (9 * m) + i * m + col];
        }
        res[row * m + col] = sum;
    }
}
```

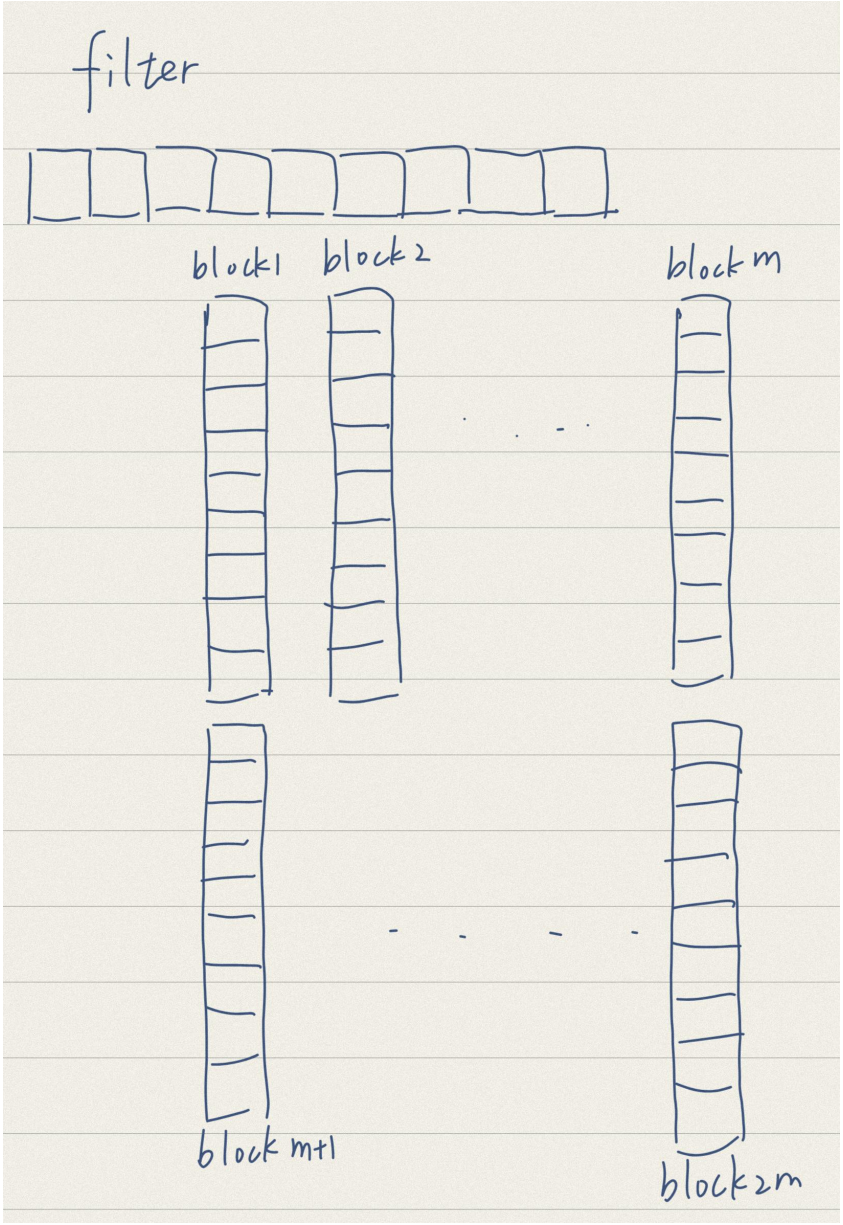
首先需要在 task3 的基础上将原始的加上填充的图像矩阵进行变形

```
// 应该包括将原图像矩阵转化成 im2col 所用的矩阵的时间
// 将每个 3x3 的方块变成一个 1x9 的列向量，按照步长逐个分解后变成新的矩阵
    for (int c = 0; c < 3; c++)
    {
        int r = 0, l = 0;
        for (int i = 0; i < n - 2; i += stride)
        {
            for (int j = 0; j < n - 2; j += stride)
            {
                im_mat[c * s_block + r * m + l] = mat[c * m_block + i * n + j];
                im_mat[c * s_block + (r + 1) * m + l] = mat[c * m_block + i * n + j + 1];
                im_mat[c * s_block + (r + 2) * m + l] = mat[c * m_block + i * n + j + 2];
                im_mat[c * s_block + (r + 3) * m + l] = mat[c * m_block + (i + 1) * n + j];
                im_mat[c * s_block + (r + 4) * m + l] = mat[c * m_block + (i + 1) * n + j + 1];
                im_mat[c * s_block + (r + 5) * m + l] = mat[c * m_block + (i + 1) * n + j + 2];
                im_mat[c * s_block + (r + 6) * m + l] = mat[c * m_block + (i + 2) * n + j];
                im_mat[c * s_block + (r + 7) * m + l] = mat[c * m_block + (i + 2) * n + j + 1];
                im_mat[c * s_block + (r + 8) * m + l] = mat[c * m_block + (i + 2) * n + j + 2];
                l += 1;
            }
            r += 9;
            l = 0;
        }
    }
```

在这里我们采用和指导文案稍微有点不同的方法，文档中将每个 3×3 的块转化成 9×1 的列向量后沿行方向叠加，最后每个 channel 的图像矩阵变成一个 9 行、m×m 列的新矩阵，在这里我将其修改成 9×

m 行、m 列的矩阵，更加符合结果矩阵的形态，方便进行计算。在卷积函数中，把结果矩阵每个点的计算分分配到一个 CUDA 线程上，使用卷积核（1×9 行向量）和该节点对应的 9×1 列向量进行矩阵乘法（内积）得到结果。

示意图如下：



编译指令：nvcc task4.cu -o task4

task5：参考文件 task5.cu

```
int main()
{
    int n;
    cin >> n;
    // 初始化 cudnn
    cudnnHandle_t handle;
    cudnnCreate(&handle);

    // 定义数据维度
```

```
int batch_size = 1;
int in_channels = 3;
int in_height = n;
int in_width = n;
int out_channels = 1;
int fileter_height = 3;
int filter_width = 3;
int stride = 3;
int padding = 1;
int m = (n + 2 * padding - fileter_height) / stride + 1;
int out_height = m;
int out_width = m;

// 定义输入输出描述符
cudnnTensorDescriptor_t input_Desc, output_Desc;
cudnnFilterDescriptor_t filter_Desc;
cudnnConvolutionDescriptor_t conv_Desc;

// 创建描述符
cudnnCreateTensorDescriptor(&input_Desc);
cudnnCreateTensorDescriptor(&output_Desc);
cudnnCreateFilterDescriptor(&filter_Desc);
cudnnCreateConvolutionDescriptor(&conv_Desc);

// 设置描述符
cudnnSetTensor4dDescriptor(input_Desc, CUDNN_TENSOR_NCHW, CUDNN_DATA_FLOAT,
batch_size, in_channels, in_height, in_width);

cudnnSetTensor4dDescriptor(output_Desc, CUDNN_TENSOR_NCHW, CUDNN_DATA_FLOAT,
batch_size, out_channels, out_height, out_width);

cudnnSetFilter4dDescriptor(filter_Desc, CUDNN_DATA_FLOAT, CUDNN_TENSOR_NCHW,
out_channels, in_channels, fileter_height, filter_width);

cudnnSetConvolution2dDescriptor(conv_Desc, padding, padding, stride, stride, 1, 1,
CUDNN_CONVOLUTION, CUDNN_DATA_FLOAT);

// 分配设备内存、
float *d_input, *d_output, *d_filter;
size_t input_size = batch_size * in_channels * in_height * in_width * sizeof(float);
size_t output_size = batch_size * out_channels * out_height * out_width * sizeof(float);
size_t filter_size = out_channels * in_channels * fileter_height * filter_width * sizeof(float);
float *input = (float*)malloc(input_size);
float *output = (float*)malloc(output_size);
random_ints(input, 3 * n * n);
float filter[27]=
{
    1.5, -2.5, 1.5,
    -2.5, 4.0, -2.5,
    1.5, -2.5, 1.5,
    1.5, -2.5, 1.5,
    -2.5, 4.0, -2.5,
    1.5, -2.5, 1.5,
    1.5, -2.5, 1.5,
    -2.5, 4.0, -2.5,
    1.5, -2.5, 1.5
};

cudaMalloc((void**)&d_input, input_size);
cudaMalloc((void**)&d_output, output_size);
cudaMalloc((void**)&d_filter, filter_size);

// 将数据拷贝到设备
cudaMemcpy(d_input, input, input_size, cudaMemcpyHostToDevice);
cudaMemcpy(d_filter, filter, filter_size, cudaMemcpyHostToDevice);

// 创建 cudnn 激活函数描述符
float alpha = 1.0f;
float beta = 0.0f;

// 记录开始时间
cudaEvent_t start, stop;
cudaEventCreate(&start);
cudaEventCreate(&stop);
cudaEventRecord(start, 0);

// 前向传播
cudnnConvolutionForward(handle, &alpha, input_Desc, d_input, filter_Desc, d_filter, conv_Desc,
CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM, NULL, 0, &beta, output_Desc, d_output);

// 记录结束时间
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);

float time;
cudaEventElapsedTime(&time, start, stop);
```

```
    // 销毁事件
    cudaEventDestroy(start);
    cudaEventDestroy(stop);

    // 将结果拷贝到主机
    cudaMemcpy(output, d_output, output_size, cudaMemcpyDeviceToHost);

    cout << "Padding: " << padding << " Stride: " << stride << endl;
    cout << "Input:" << endl;
    for (int i = 0; i < min(n, 5); i++)
    {
        for (int c = 0; c < 3; c++)
        {
            for (int j = 0; j < min(n, 5); j++)
            {
                cout << setw(8) << input[c * n * n + i * n + j] << " ";
            }
            if (c != 2)
            {
                cout << "|";
            }
        }
        cout << endl;
    }

    cout << "Result:" << endl;
    for (int i = 0; i < min(5, m); i++)
    {
        for (int j = 0; j < min(5, m); j++)
        {
            cout << setw(8) << output[i * m + j] << " ";
        }
        cout << endl;
    }
    cout << "Used time: " << time << "ms" << endl;

    // 释放内存
    cudaFree(d_input);
    cudaFree(d_output);
    cudaFree(d_filter);


    free(input);
    free(output);

    // 销毁描述符
    cudnnDestroyTensorDescriptor(input_Desc);
    cudnnDestroyTensorDescriptor(output_Desc);
    cudnnDestroyFilterDescriptor(filter_Desc);
    cudnnDestroyConvolutionDescriptor(conv_Desc);
    cudnnDestroy(handle);

    return 0;
}
```

首先需要创建 cudnn 句柄，并为所需的参数赋初值，初始化后需要创建输入矩阵、输出矩阵、卷积核以及卷积运算的描述符，需要注意的是即使 3 个通道使用同一个卷积核也要把卷积核重复 3 遍进行扩展，通过 cudnnConvolutionForward 前向传播函数来进行卷积运算，计算结束后需要销毁描述符和句柄。

编译指令：nvcc task5.cu -o task5 -lcudnn -lcudart -lcuda

3. 实验结果

task1:

个人电脑：

Block_size	Order of Matrix (milliseconds)				
	512	1024	2048	4096	8192
32	8.34333	66.0152	405.196	2798.97	23963.7
64	4.37606	33.9856	227.96	1584.41	13553.2
128	2.37792	18.0659	136.538	1003.68	8696.78
256	1.3943	10.093	79.714	590.115	4751.72
512	1.09549	8.29821	65.5219	456.113	3848.87

通过打印部分值来验证代码实现的准确性：

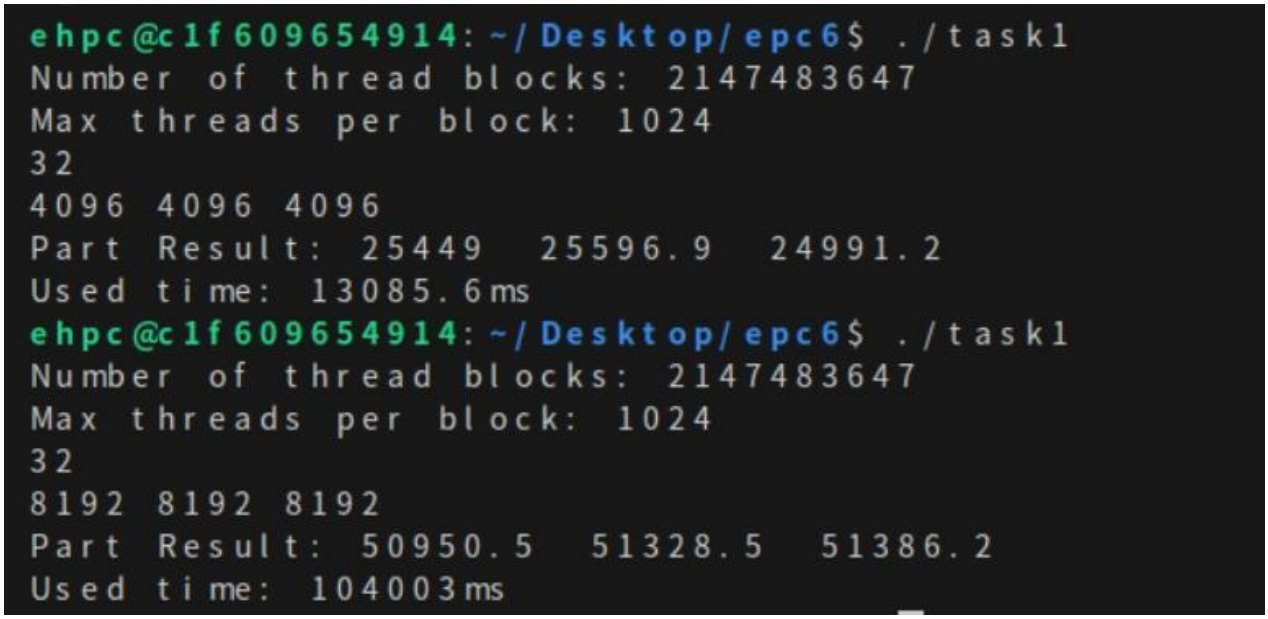
```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task1
Number of thread blocks: 2147483647
Max threads per block: 1024
2048 2048 2048
Part Result: 12984.8  12813.6  12814.5
Used time: 405.196ms
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task1
Number of thread blocks: 2147483647
Max threads per block: 1024
4096 4096 4096
Part Result: 25472.5  25887.7  25915.2
Used time: 2798.97ms
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task1
Number of thread blocks: 2147483647
Max threads per block: 1024
8192 8192 8192
Part Result: 50936.8  50743.2  51888.9
Used time: 23963.7ms
```

实验平台：

Block_size	Order of Matrix (milliseconds)				
	512	1024	2048	4096	8192

32	39.3427	296.521	1768.4	13085.6	104003
64	20.3088	160.261	984.784	6877.41	55476.1
128	10.9311	85.4603	593.625	3754.77	30327.4
256	6.3673	49.7884	359.398	2238.74	16898.9
512	4.16726	32.5379	256.621	1550.78	11071.1

打印部分值：



可以观察到当线程块的大小（Block_size）增大时，虽然使用的总线程数不变，但是用到的线程块数量减少，在每个线程块之间的共享内存能够加快运算。

task2:

Order of Matrix (milliseconds)				
512	1024	2048	4096	8192
0.220032	0.893376	5.83936	40.5932	370.317

部分结果：

```
ehpc@c1f609654914:~/Desktop/epc6$ ./task2
4096 4096 4096
Part Result: 26457  25252.4  25984.7
Used time: 40.5337ms
ehpc@c1f609654914:~/Desktop/epc6$ ./task2
8192 8192 8192
Part Result: 52276.9  51209.8  52153
Used time: 370.317ms
```

使用基于 cublas 的矩阵乘法，可以很大程度上提高矩阵运算的性能。较自己手动分配线程块以及 SIMD 线程有接近 10 倍的提升，可以通过增加 BLOCK_SIZE 至 1024（可以分配成 32×32），利用 shared memory 等效于主机系统中的 cache 的优势进行优化，能够较大程度上提高性能。

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task1
Number of thread blocks: 2147483647
Max threads per block: 1024
8192 8192 8192
Part Result: 50987.6  51181.6  51454.5
Used time: 3839.18ms
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task1_1
Number of thread blocks: 2147483647
Max threads per block: 1024
8192 8192 8192
Part Result: 51864.6  50545.9  51550.3
Used time: 2657.84ms
```

```
ehpc@605df6d63e61:~/Desktop/hpc$ ./task1_1
Number of thread blocks: 2147483647
Max threads per block: 1024
8192 8192 8192
Part Result: 51076.4  51566.8  51310.7
Used time: 3362.35ms
ehpc@605df6d63e61:~/Desktop/hpc$ ./task1
Number of thread blocks: 2147483647
Max threads per block: 1024
8192 8192 8192
Part Result: 51167.1  51908.8  50953.2
Used time: 8177.41ms
```

在实验平台上提升效果更好。

task3:（仅打印前 5×5 作为验证）

卷积核：

[1.5　-2.5　1.5

-2.5　4　-2.5

1.5　-2.5　1.5]

stride=1，padding = 1：

Order of Matrix (milliseconds)				
256	512	1024	2048	4096
0.050272	0.098272	0.200896	0.592448	2.20032

验证：m=(n+2*padding-3)/stride+1=5,填充后输入输出大小一致。

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task3
5
Padding: 1 Stride: 1
Input:
   207.9      264     23.1    182.6    212.3 |     9.9     37.4     85.8    228.8    185.9 |    69.3     13.2    246.4    138.6    267.3
    38.5     47.3     35.2      9.9     57.2 |   179.3       33    126.5    172.7    239.8 |   128.7    270.6    279.4    161.7      7.7
   256.3     59.4    193.6      187    255.2 |   125.4     71.5    171.6     50.6      143 |   191.4    160.6    102.3      176     15.4
   250.8    157.3     80.3    224.4    144.1 |   190.3      143       44     46.2    117.7 |   251.9    248.6      9.9    251.9    268.4
     7.7     82.5    265.1    194.7    277.2 |    35.2       66    170.5     70.4     79.2 |   133.1    234.3    147.4     80.3    174.9
Result:
  194.15    933.9 -1063.15   140.25    654.5
 -1251.8   443.85   323.95 -43.4501   -894.3
  1146.2  -1262.8   1318.9  -600.05  -166.65
 -109.45  1141.25    -2618   1524.6   -409.2
 -650.65  -497.75   1783.1  -1713.8   719.95
Used time: 0.024576ms
```

取结果矩阵（1，1）位置的 443.85 进行验证。

[207.9　264　23.1　　[9.9　37.4　85.8　　[69.3　13.2　246.4

38.5　47.3　35.2　　179.3　33　126.5　　128.7　270.6　279.4

256.3　59.4　193.6]　　125.4　71.5　171.6]　　191.4　160.6　102.3]

(207.9*1.5-264*2.5+23.1*1.5-38.5*2.5+47.3*4-35.2*2.5+256.3*1.5-59.4*2.5+193.6*1.5)+(9.9*1.5-37.4*2.5+85.8*1.5-179.3*2.5+33*4-126.5*2.5+125.4*1.5-71.5*2.5+171.6*1.5)+(69.3*1.5-13.2*2.5+246.4*1.5-128.7*2.5+270.6*4-279.4*2.5+191.4*1.5-160.6*2.5+102.3*1.5)=443.85

经验证，以上的结果正确。

stride=2，padding = 1：

Order of Matrix (milliseconds)				
256	512	1024	2048	4096
0.067744	0.078176	0.14944	0.420032	1.37341

验证 m=(n+2*padding-3)/stride+1=3，5×5->3×3

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task3
5
Padding: 1 Stride: 2
Input:
   214.5    246.4    177.1    238.7    269.5 |   196.9    247.5    258.5    113.3    105.6 |   130.9    183.7     53.9    169.4      165
   106.7    212.3    214.5    266.2     52.8 |   180.4     36.3    133.1     47.3      165 |   192.5    135.3    106.7      165    135.3
    31.9    192.5     85.8    183.7    266.2 |   140.8      7.7    111.1    265.1    159.5 |   180.4    206.8    191.4    161.7    272.8
    75.9    199.1     36.3    174.9    162.8 |   106.7     70.4    113.3    202.4    174.9 |   189.2      198     19.8      121     30.8
   114.4    258.5    151.8     37.4    254.1 |    220    224.4    188.1     79.2       66 |   245.3    169.4       99      143     92.4
Result:
-1256.75  -634.15    589.6
 -255.75    192.5   928.95
   388.3   501.05    827.2
Used time: 0.024576ms
```

同上面的方法，容易验证卷积结果正确。

stride=3，padding = 1：

Order of Matrix (milliseconds)				
256	512	1024	2048	4096
0.07072	0.08112	0.150112	0.383296	1.31597

验证：m=(n+2*padding-3)/stride+1=2,5×5->2×2

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task3
5
Padding: 1 Stride: 3
Input:
    83.6      275    146.3      7.7      187 |   102.3    213.4     28.6    225.5      154 |   193.6    257.4     14.3    268.4    162.8
   238.7    205.7      209    205.7    251.9 |    30.8    211.2    250.8     20.9    228.8 |   259.6     90.2    152.9    103.4       33
   228.8      6.6    118.8    145.2    191.4 |   245.3    240.9      143     74.8     12.1 |     1.1    203.5    123.2    151.8     20.9
    34.1    130.9    108.9    112.2    128.7 |     6.6    245.3     50.6    246.4    178.2 |    19.8    136.4      176     20.9      8.8
   166.1     30.8     49.5     57.2     53.9 |    62.7     93.5    111.1    277.2    246.4 |    82.5      110       22    149.6     53.9
Result:
 -1348.6  2032.25
-1846.35   -530.2
Used time: 0.026624ms
```

同上面的方法，容易验证卷积结果正确。

task4:（仅打印前 5×5 作为验证，卷积核同 task3）

stride=1，padding = 1：

Order of Matrix (milliseconds)				
256	512	1024	2048	4096
8.448	39.2387	157.705	578.762	2512

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task4
5
Padding: 1 Stride: 1
Input:
   194.7    215.6    155.1    255.2    117.7 |   137.5    243.1    214.5    229.9    169.4 |   240.9    214.5    128.7    247.5    262.9
   147.4     13.2     29.7     85.8    163.9 |     187    201.3    218.9    226.6     59.4 |   130.9     35.2    155.1     17.6    123.2
   195.8    136.4    115.5    212.3     89.1 |     253    278.3    170.5    129.8    108.9 |    79.2      242    237.6    172.7      253
   249.7     20.9    100.1      275    149.6 |    71.5    193.6     35.2     92.4     69.3 |    67.1    114.4    122.1    102.3     36.3
   123.2    177.1    124.3    273.9    216.7 |   130.9    267.3    113.3    206.8    200.2 |   257.4    188.1    260.7    183.7     68.2
First 5x5 changed_mat:
       0        0        0        0        0 |       0        0        0        0        0 |       0        0        0        0        0
       0        0        0        0        0 |       0        0        0        0        0 |       0        0        0        0        0
       0        0        0        0        0 |       0        0        0        0        0 |       0        0        0        0        0
       0    194.7    215.6    155.1    255.2 |       0    137.5    243.1    214.5    229.9 |       0    240.9    214.5    128.7    247.5
   194.7    215.6    155.1    255.2    117.7 |   137.5    243.1    214.5    229.9    169.4 |   240.9    214.5    128.7    247.5    262.9
   215.6    155.1    255.2    117.7        0 |   243.1    214.5    229.9    169.4        0 |   214.5    128.7    247.5    262.9        0
       0    147.4     13.2     29.7     85.8 |       0      187    201.3    218.9    226.6 |       0    130.9     35.2    155.1     17.6
   147.4     13.2     29.7     85.8    163.9 |     187    201.3    218.9    226.6     59.4 |   130.9     35.2    155.1     17.6    123.2
    13.2     29.7     85.8    163.9        0 |   201.3    218.9    226.6     59.4        0 |    35.2    155.1     17.6    123.2        0
Result:
  -179.3   693.55    -1661   609.95 -2.74993
  479.05 -1313.95  1476.75  -639.65 -70.4001
 -795.85   823.35  -419.65  -482.35    212.3
      66  -429.55   180.95    586.3   -724.9
  -12.65    151.8  -694.65  -206.25   345.95
Used time: 0.208896ms
```

其中 changed_mat 表示将矩阵转化成使用 im2col 方法的矩阵格式,经验证，卷积的结果正确。

stride=2，padding = 1：

Order of Matrix (milliseconds)				
256	512	1024	2048	4096
4.05504	8.99994	32.0707	126.582	493.324

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task4
5
Padding: 1 Stride: 2
Input:
    40.7       77    144.1     84.7    126.5 |   149.6    108.9    229.9      132    257.4 |   159.5     49.5    273.9    191.4     23.1
   271.7    228.8    118.8    205.7     35.2 |   269.5     31.9    280.5    118.8    247.5 |   152.9    127.6     84.7     36.3    277.2
     154    133.1      132     57.2     20.9 |    67.1     14.3    140.8      198    211.2 |   210.1    147.4    118.8      9.9    222.2
    79.2      209     64.9       44    239.8 |    52.8     23.1    114.4    115.5    101.2 |   136.4     94.6    258.5     78.1     64.9
   181.5    138.6    137.5    244.2     57.2 |   177.1       33      2.2    189.2     69.3 |   205.7    213.4    223.3     70.4     97.9
First 5x5 changed_mat:
       0        0        0        0        0 |       0        0        0        0        0 |       0        0        0        0        0
       0        0        0        0        0 |       0        0        0        0        0 |       0        0        0        0        0
       0        0        0        0       77 |       0        0        0        0    108.9 |       0        0        0        0     49.5
       0       77     84.7     40.7    144.1 |       0    108.9      132    149.6    229.9 |       0     49.5    191.4    159.5    273.9
    40.7    144.1    126.5       77     84.7 |   149.6    229.9    257.4    108.9      132 |   159.5    273.9     23.1     49.5    191.4
      77     84.7        0        0    228.8 |   108.9      132        0        0     31.9 |    49.5    191.4        0        0    127.6
       0    228.8    205.7    271.7    118.8 |       0     31.9    118.8    269.5    280.5 |       0    127.6     36.3    152.9     84.7
   271.7    118.8     35.2    228.8    205.7 |   269.5    280.5    247.5     31.9    118.8 |   152.9     84.7    277.2    127.6     36.3
   228.8    205.7        0        0    228.8 |    31.9    118.8        0        0     31.9 |   127.6     36.3        0        0    127.6
Result:
  -342.1    896.5   -250.8
 -345.95  -167.75  -362.45
 1113.75 -1018.05 -1020.25
Used time: 0.23552ms
```

通过计算可以验证结果正确性。

stride=3，padding = 1：

Order of Matrix (milliseconds)				
256	512	1024	2048	4096

1.24211	4.06733	14.6166	56.4009	215.052

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task4
5
Padding: 1 Stride: 3
Input:
   102.3     67.1    203.5    169.4    162.8 |   196.9     14.3      110    211.2     78.1 |   148.5    113.3    244.2    206.8    172.7
   257.4     64.9     69.3    178.2    130.9 |   221.1     48.4    234.3    118.8       77 |   114.4     86.9    195.8    226.6      231
     3.3    279.4    125.4    233.2     84.7 |   248.6    149.6    181.5    254.1    272.8 |   261.8    174.9    254.1    195.8     47.3
   258.5    246.4    205.7    182.6    244.2 |     2.2    108.9     69.3    246.4     47.3 |   162.8    188.1    185.9    203.5    160.6
   201.3     67.1       22    189.2      121 |   204.6    246.4    149.6    223.3    161.7 |     6.6    261.8     15.4    119.9    125.4
First 5x5 changed_mat:
       0        0        0        0        0 |       0        0        0        0        0 |       0        0        0        0        0
       0        0        0        0        0 |       0        0        0        0        0 |       0        0        0        0        0
       0        0        0    203.5    102.3 |       0        0        0      110    196.9 |       0        0        0    244.2    148.5
       0    203.5    102.3    169.4     67.1 |       0      110    196.9    211.2     14.3 |       0    244.2    148.5    206.8    113.3
   102.3    169.4     67.1    162.8        0 |   196.9    211.2     14.3     78.1        0 |   148.5    206.8    113.3    172.7        0
    67.1    162.8        0     69.3    257.4 |    14.3     78.1        0    234.3    221.1 |   113.3    172.7        0    195.8    114.4
       0     69.3    257.4    178.2     64.9 |       0    234.3    221.1    118.8     48.4 |       0    195.8    114.4    226.6     86.9
   257.4    178.2     64.9    130.9        0 |   221.1    118.8     48.4       77        0 |   114.4    226.6     86.9      231        0
    64.9    130.9        0    125.4      3.3 |    48.4       77        0    181.5    248.6 |    86.9      231        0    254.1    261.8
Result:
   122.1     19.8
  -211.2   -449.9
Used time: 0.144384ms
```

通过计算可以验证结果正确性。

随着矩阵规模增大，消耗的时间大致成线性关系。

与 task3 使用的卷积运算方法相比，im2col 方法需要一个额外的将原始图像矩阵的一个 3×3 形状转变成 1×9 的形状，这一步在 CPU 上进行，因此需要 O(m^2)的时间复杂度(m^2 为结果矩阵的大小)，因此所需要的时间显著增加。

task5:（仅打印前 5×5 作为验证，卷积核同 task3）

stride=1，padding = 1：

Order of Matrix (milliseconds)				
256	512	1024	2048	4096
873.795	933.911	882.647	838.613	922.555

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task5
5
Padding: 1 Stride: 1
Input:
   119.9    278.3    204.6    167.2     30.8 |   123.2     27.5       11     45.1    278.3 |   174.9    217.8       33    183.7    203.5
   204.6    212.3    229.9     49.5    158.4 |   261.8     96.8    168.3    266.2      264 |    95.7    190.3     40.7     52.8    211.2
      22     39.6     45.1     17.6    115.5 |   177.1    276.1      110    251.9    212.3 |   258.5    247.5    113.3    277.2     30.8
   234.3    247.5    181.5    233.2    232.1 |   246.4     86.9    119.9    183.7    169.4 |   244.2     61.6      3.3      242     36.3
    84.7     35.2     36.3      209    117.7 |   237.6     31.9     36.3     83.6    246.4 |   188.1     30.8      2.2     73.7    128.7
Result:
 -293.15    680.9    -1100    368.5    29.15
   441.1    -1133   1337.6  -1481.7    849.2
   -1452   1245.2  -675.95    773.3 -1069.75
  481.25  -675.95     84.7   288.75  -655.05
   578.6     -517    -40.7   -487.3    949.3
Used time: 909.342ms
```

通过前几个数据可以验证结果的正确性。

stride=2，padding = 1：

Order of Matrix (milliseconds)				
256	512	1024	2048	4096
729.968	749.472	717.98	727.661	788.575

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task5
5
Padding: 1 Stride: 2
Input:
    42.9    138.6    148.5     56.1    255.2 |    67.1    113.3    217.8    216.7    201.3 |     220    249.7    128.7     92.4     35.2
    28.6      5.5    212.3    226.6    229.9 |      44     56.1    150.7     38.5      209 |   188.1      132     83.6    182.6    236.5
     143     80.3     29.7    180.4    166.1 |    31.9      165    226.6     56.1     96.8 |   140.8      209    125.4    269.5     97.9
   184.8     59.4    271.7       44     19.8 |   160.6      9.9      154       77    160.6 |   147.4    254.1    130.9      231    238.7
     176    169.4     85.8     94.6    146.3 |   259.6    239.8    163.9     78.1    213.4 |   173.8    260.7    171.6      187    261.8
Result:
 -295.35  -341.55    36.85
  -981.2  -1406.9  -1358.5
   15.95  -1267.2     1067
Used time: 747.517ms
```

通过计算可以验证结果正确性

stride=3，padding = 1：

Order of Matrix (milliseconds)				
256	512	1024	2048	4096
753.432	830.133	842.914	792.121	812.337

```
hzc_hho@hzc-hho:/mnt/c/Users/陈圳煌/Desktop/计算机学院/计算机学院（大三上）/高性能程序设计/test6$ ./task5
5
Padding: 1 Stride: 3
Input:
     132    272.8    244.2      187     23.1 |   227.7    145.2    111.1    280.5    185.9 |   228.8    184.8     90.2      198    101.2
   278.3    103.4    137.5    199.1    170.5 |    42.9       22    222.2      220    156.2 |   237.6    117.7    134.2    116.6    247.5
   137.5    225.5    122.1    213.4    126.5 |   113.3     84.7      110    108.9    115.5 |    31.9    222.2     38.5    113.3    125.4
   228.8     93.5    101.2    147.4    174.9 |    94.6    260.7    106.7    280.5    206.8 |   83.6    265.1    172.7       33     63.8
   256.3     75.9      275     19.8    277.2 |    94.6     97.9       22    126.5    251.9 |   47.3    205.7    173.8     45.1     15.4
Result:
 -185.35  1035.65
 -254.65    690.8
Used time: 762.975ms
```

通过计算可以验证结果正确性

与 task3 中的结果相比性能慢了很多，推测可能的原因：个人计算机硬件配置和安装的 cuDNN 库的优化匹配不当；观察到随着矩阵的维度增加，自己实现的卷积运算大致有线性的增长趋势，但是基于 cuDNN 的卷积运算没有很大变化，推测数据的 Batch_size 以及矩阵规模太小，cuDNN 不能完全利用 GPU 并行化的能力。

4. 实验感想

这次的实验是我第一次接触 CUDA 编程，从最开始的对 GPU 的结构还不了解，完全不知道从什么地方开始入手，经过老师的讲解和查阅资料后能够初步地使用 GPU 资源进行一些简单的编程，也能够较为顺利地完成了实验任务，感觉还是受益匪浅的。经过这学期的课程学习，我对多种并行方法都有了一些了解和初步使用的能力，也从一次次实验中收获了进步和对并行这方面的兴趣，感谢老师和助教在这一学期的时间内对高性能计算程序设计这门课程上给予的指导。