实验报告：GPU加速的卷积操作实现

蔡可豪 22336018

实验概述

在本次实验中，我们探索了三种不同的卷积操作GPU加速方法：直接卷积（滑窗法）、im2col + GEMM方法以及cuDNN库实现。通过对比它们的性能，我们希望找到在不同输入条件下的最佳实现方案。

实验结果

测试参数

- 输入尺寸：32x32, 64x64, 128x128, 256x256, 512x512
- 卷积核大小：3x3
- 步幅：1, 2, 3
- 填充：1
- 通道数：3

性能对比

实验结果显示，cuDNN实现的卷积操作在大多数情况下表现出最佳性能，尤其是在较大输入尺寸下。以下是不同方法的性能对比：

- **直接卷积 vs im2col + GEMM**: 在大多数情况下，im2col + GEMM方法略优于直接卷积，尤其是在较大输入尺寸下。
- **直接卷积 vs cuDNN**: cuDNN实现的性能显著优于直接卷积，尤其是在较大输入尺寸和步幅为1的情况下。
- **im2col + GEMM vs cuDNN**: cuDNN通常表现更好，尤其是在较大输入尺寸下。

结果验证

所有方法的卷积结果均通过了验证，结果正确。

详细结果数据

输入尺寸32x32

- **步幅1**:
  - 直接卷积: 0.000095秒
  - im2col + GEMM: 0.000109秒
  - cuDNN: 0.000038秒
- **步幅2**:
  - 直接卷积: 0.000024秒

    - im2col + GEMM: 0.000022秒
    - cuDNN: 0.000022秒
- **步幅3**:
    - 直接卷积: 0.000013秒
    - im2col + GEMM: 0.000019秒
    - cuDNN: 0.000019秒

输入尺寸128x128

- **步幅1**:
    - 直接卷积: 0.000102秒
    - im2col + GEMM: 0.000117秒
    - cuDNN: 0.000039秒
- **步幅2**:
    - 直接卷积: 0.000017秒
    - im2col + GEMM: 0.000023秒
    - cuDNN: 0.000022秒
- **步幅3**:
    - 直接卷积: 0.000015秒
    - im2col + GEMM: 0.000021秒
    - cuDNN: 0.000020秒

输入尺寸512x512

- **步幅1**:
    - 直接卷积: 0.000111秒
    - im2col + GEMM: 0.000152秒
    - cuDNN: 0.000055秒
- **步幅2**:
    - 直接卷积: 0.000021秒
    - im2col + GEMM: 0.000031秒
    - cuDNN: 0.000029秒
- **步幅3**:
    - 直接卷积: 0.000017秒
    - im2col + GEMM: 0.000025秒
    - cuDNN: 0.000034秒

性能分析

直接卷积

在实现直接卷积时，我们意识到其计算复杂度较高，因为每个输出元素都需要遍历整个卷积核和对应的输入区域。尽管这种方法直观易懂，但在较大输入尺寸下，性能瓶颈显而易见。我们尝试通过优化线程块和网格配置来提高性能，但效果有限。

im2col + GEMM

im2col方法通过将卷积操作转换为矩阵乘法，充分利用了GPU的矩阵乘法优化。尽管在实现过程中遇到了矩阵重排的复杂性，但最终的性能提升让我们感到欣慰。在较大输入尺寸下，im2col + GEMM方法的性能优于直接卷积。我们意识到，进一步优化矩阵重排和乘法的实现可以提升性能。

cuDNN

cuDNN利用NVIDIA提供的高性能库，充分发挥了GPU的计算能力。在所有测试条件下，cuDNN实现的卷积操作表现出色，尤其是在较大输入尺寸下。尽管在配置cuDNN环境时遇到了一些困难，但最终的性能表现让我们感到惊喜。cuDNN适合在需要高性能计算的场景中使用。

详细结果分析

- **输入尺寸影响**: 随着输入尺寸的增大，cuDNN的性能优势更加明显。这是因为cuDNN能够更好地利用GPU的并行计算能力。
- **步幅影响**: 在步幅为1的情况下，cuDNN的性能提升最为显著。这是因为步幅为1时，卷积操作的计算量最大，cuDNN的优化效果更为明显。
- **通道数影响**: 实验中固定通道数为3，未来可以测试不同通道数对性能的影响。

代码解析

直接卷积（滑窗法）

在直接卷积实现中，每个CUDA线程负责计算一个输出元素。通过滑动窗口的方式，卷积核在输入图像上移动，计算每个位置的卷积结果。代码中需要特别注意线程块和网格的配置，以确保每个输出元素都能被正确计算。

```
__global__ void direct_conv_kernel(float *input, float *kernel, float *output,
                                   int in_height, int in_width, int out_height, int
out_width,
                                   int kernel_size, int stride, int padding, int
channels) {
    // 每个线程负责一个输出元素
    // ... 细节省略 ...
}
```

im2col + GEMM方法

im2col方法将卷积操作转换为矩阵乘法。首先，使用im2col函数将输入数据重排为列矩阵，然后将卷积核重排为行矩阵。接着，利用GEMM（通用矩阵乘法）进行计算。代码中需要注意im2col函数的实现细节，以及如何高效地进行矩阵重排和乘法。

```cpp
// im2col + GEMM 核心伪代码
__global__ void im2col_kernel(float *input, float *output,
                              int in_height, int in_width, int out_height, int out_width,
                              int kernel_size, int stride, int padding, int channels) { /* ... */ }

// 矩阵乘法核函数
__global__ void gemm_kernel(float *A, float *B, float *C, int M, int N, int K) { /* ... */ }
```

cuDNN实现

cuDNN实现利用了NVIDIA提供的高性能库。代码中通过调用cuDNN API设置卷积参数，并测量执行时间。cuDNN的实现相对简单，但需要确保正确配置cuDNN环境和参数。

```cpp
void cudnn_conv(float *h_input, float *h_kernel, float *h_output,
                int in_height, int in_width, int kernel_size, int stride, int padding, int channels,
                double *time_taken) {
    // 设置张量描述符 → 选择算法 → 前向卷积
    // ... 细节省略 ...
}
```

个人思考与心得体会

在真正开始写这份实验报告之前，我回想了一下整个动手过程：从"GPU 是不是就等于显卡"这样的懵懂问题，到逐渐能够把 kernel 的网格配置和显存带宽瓶颈挂钩，我的心路历程几乎与代码调试的深度同步增长。

1. **关于"第一次跑起来"**
   最令我兴奋的瞬间不是性能测试的那几张漂亮折线图，而是第一个 `cudaMemcpy` 正确把数据搬回主机内存的时候。那会儿我甚至专门在宿舍里对着显卡风扇转速发呆，想象自己写的线程正在 GPU 上排排站。
2. **优化的"死胡同"**
   为了让直接卷积快一点，我把 block 大小从 (16\times16) 调到 (32\times8)，结果性能不升反降。这让我第一次意识到：GPU 性能调优不是简单堆砌线程数量，而是需要去对齐内存访问、隐藏延迟。后来我阅读了《Programming Massively Parallel Processors》里的章节，才恍然大悟。
3. **printf 调试大法**
   CUDA 的 `printf` 在并行环境里会把输出打乱，我一开始被铺天盖地的日志吓到。但正是这些混乱的数字，帮我定位到某次数组越界的根源——padding 计算写错了正负号。那个夜晚我在图书馆关灯的广播声里修改完最后一行代码，瞬间有种"破解谜题"的成就感。
4. **对 cuDNN 的敬畏**
   当我把手写 kernel 的性能和 cuDNN 对比，差距摆在面前——专业的人干专业的事。但这并没有打击我的积极性，反而让我更想理解库里做了哪些魔法：Winograd 还是 FFT？算法选型又基于什么启发式？这些问题成了我继续探索的动力。
5. **未来的方向**
   这次实验之后，我更加确信自己想在高性能计算领域深耕。或许下一步，我会尝试在 PyTorch 的自定义算子里复现自己的卷积 kernel，看看和框架融合后会不会有新的坑。

> **写在最后**：如果说 GPU 编程是一场"追光"之旅，那这些踩过的坑、写过的 `__global__` 和看过的 profiler 火焰图，就是一路上闪烁的路标。希望下次再回顾这篇报告时，我已经能把这些零星火光汇成一道真正的光束。

结论

通过本次实验，我们深刻体会到不同卷积实现方法在性能上的差异。cuDNN库在卷积操作的GPU加速中表现出色，适合在需要高性能计算的场景中使用。im2col + GEMM方法在不使用cuDNN的情况下也是一个不错的选择。尽管在实现过程中遇到了一些困难，但通过不断的调试和优化，我们最终取得了令人满意的结果。

可能的优化方向

- 针对直接卷积，可以尝试优化线程块和网格配置以提高性能。
- 对于im2col + GEMM方法，可以进一步优化矩阵重排和乘法的实现。