Convolution 2D

JIALI CLAUDIO, HUANG Personal code: 11032111
Peng, Rao Personal code: 11022931

Experimental setup

We use colab to run the code, and the GPU is Tesla T4. We use the following experimental configuration parameters:

Table 1: **configuration parameters**

The matrix size	8192×8192
Convolution kernel size	5×5

Divergence analysis

The divergence of the code is mainly caused by the conditional statement in the code, specifically the ghost cells. The cost of control divergence will depend on the value of width and height of the input array and the radius of the filter. Since convolution is often applied to large images, we suppose the effect of control divergence is not significant.

Constant memory and caching

We notice that the convolution filter has three interesting properties:

- The size of filter is typically small, the radius of most convolution filters is 7 or smaller.
- The filter is constant, it does not change during the convolution process.
- All the threads access the same filter.

These three properties make the filter an excellent candidate for constant memory and caching. CUDA allows us to declare variables to reside in the constant memory. Because the CUDA runtime knows that constant memory variables are not modified during kernel execution, it directs the hardware to aggressively cache the constant memory variables during kernel execution.

As a small array, the filter can be declared as a constant memory array, we use this method to accelerate the calculation, and the results are as follows:

Table 2: **The performance of the convolution with constant memory**

Optimization	Runtime(ms)
The basic 2D convolution	14.007
Using constant memory	12.443

Tiled convolution with halo cells

First, all threads in a block load the input tile and the halo cells into the shared memory, and then each thread calculates the output element. The main idea of the tiled convolution with halo can be illustrated as follows:

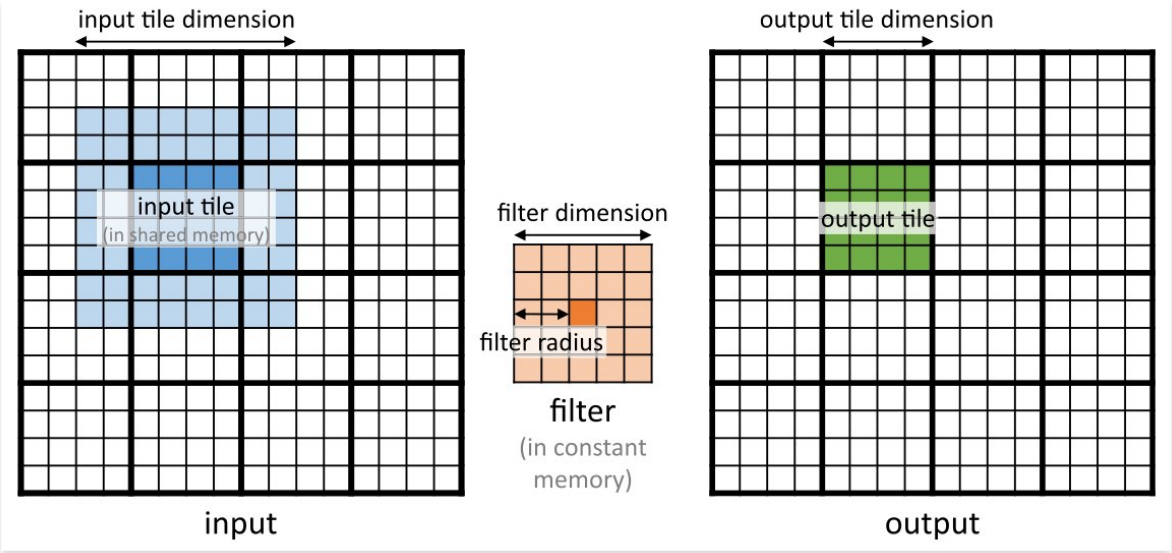

Figure 1: Input tile with halo and output tile

We can launch thread blocks whose dimension matches that of the input tiles, and each thread needs to load just one input element. The drawback of this method is that the block dimension is larger than that of the output tile, some of the threads need to be disabled during the calculation of output elements.

We choose different tile sizes to test the performance of the tiled convolution with halo cells, they are 8, 16, and 32. Tile sizes of 8, 16, and 32 are chosen because they are powers of two. This choice optimizes memory alignment and leverages GPU hardware features such as efficient thread block sizing and shared memory access patterns.

- **Small Tile (8x8)**:
  - **Advantages**: Utilizes less shared memory per block, allowing more thread blocks to reside on each Streaming Multiprocessor (SM) and increasing overall parallelism.
  - **Disadvantages**: Smaller tiles may lead to lower data reuse and higher relative overhead for boundary handling and synchronization.
- **Medium Tile (16x16)**:
  - **Advantages**: Strikes a balance between shared memory usage and data reuse. Efficiently loads and reuses data while maintaining high parallelism.
  - **Disadvantages**: May not fully utilize the available shared memory, potentially limiting performance gains.
- **Large Tile (32x32)**:
  - **Advantages**: Enhances data reuse by covering larger data blocks, potentially reducing the number of global memory accesses.
  - **Disadvantages**: Significantly increases shared memory usage per block, which can limit the number of concurrent blocks per SM and reduce overall parallelism.

The results are as follows:

Table 3: **The performance of tiled convolution with halo cells**

Optimization	Runtime (ms)
The basic 2D convolution	14.007
Using constant memory	12.443
Tiled convolution with halo cells (tile = 8)	9.489
Tiled convolution with halo cells (tile = 16)	8.521
Tiled convolution with halo cells (tile = 32)	9.491

The analysis of the results is as follows:

- **Tile Size 16**: Demonstrates the best runtime due to its optimal balance between shared memory usage and the number of thread blocks that can run concurrently on each SM. It ensures efficient data reuse without excessively limiting parallelism.
- **Tile Size 8**: Although it allows for higher parallelism, the smaller tile size results in less efficient data reuse, leading to increased memory access overhead.
- **Tile Size 32**: While larger tiles improve data reuse, the high shared memory demand restricts the number of active blocks per SM, thereby reducing overall parallel execution and negatively impacting performance.

Bibliography

[1] Wen mei W. Hwu, David B. Kirk, and Izzat El Hajj. Chapter 7 - convolution: An introduction to constant memory and caching. In Wen mei W. Hwu, David B. Kirk, and Izzat El Hajj, editors, *Programming Massively Parallel Processors (Fourth Edition)*, pages 151–171. Morgan Kaufmann, fourth edition edition, 2023.