Convolution Kernel

Definition of 2D Convolution

Given:

a) 2D image $I(\text{row}, \text{col})$ of size $\text{height} \times \text{weight}$, where $\text{col} \in 0...\text{width - 1}, \text{row} \in 0...\text{height - 1}$.
b) A filter kernel F with radius R and size $(2R+1) \times (2R+1)$.

Assume:

a) Zero-padding is applied when necessary.
b) The kernel is centered.
c) The kernel size is square.

Then the convolution operation is defined as:

$$(I * F)_{(\text{row, col})} = \sum_{\text{fRow}=0}^{2R} \sum_{\text{fCol}=0}^{2R} F_{(\text{fRow, fCol})} \cdot I_{(\text{row - R + fRow, col - R + fCol})}$$

A Basic 2D Tiled Kernel

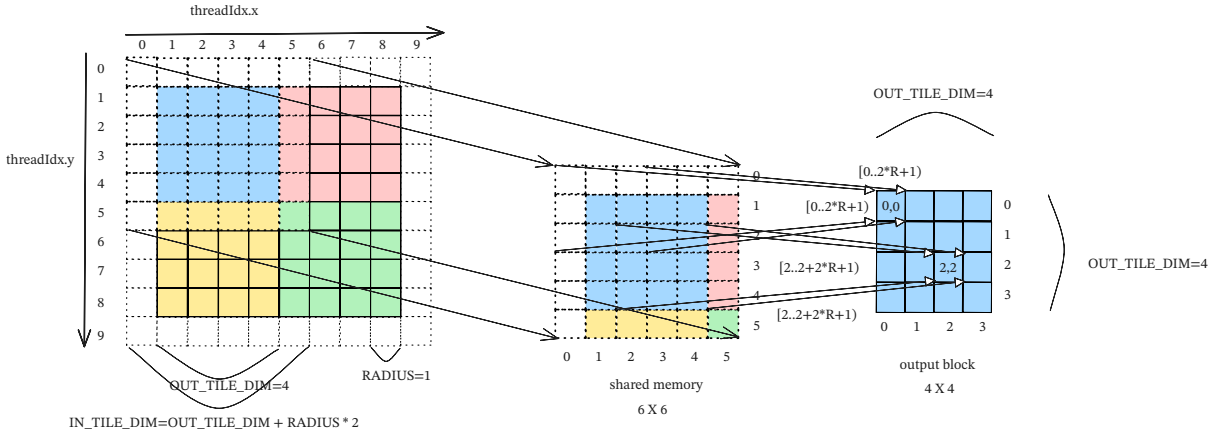

Figure 1: Visualization of A 2D Tiled Kernel.

In this kernel, each thread computes one output pixel by applying the convolution kernel over a patch of the input image.

The core idea of optimization is to reduce the performance impact of DRAM bandwidth by loading computational data into the L1 cache(shared memory) in advance, i.e. tiling.

We can decompose our approach into three steps:

a) Load data from global memory into shared memory.
b) Perform computations using shared memory and store intermediate results in registers.
c) Write the computed results from registers back to global memory.

Figure 1 explains the three steps described above. In particular, `IN_TILE_DIM` represents the number of threads per block. It is equal to `OUT_TILE_DIM + 2 * RADIUS`, where `OUT_TILE_DIM` is the number of threads responsible for computing the results and writing them back to global memory.

As shown in the figure, assume that initially, the mapping from thread to data element is defined as $\text{col} = \text{threadIdx.x} + \text{IN\_TILE\_DIM} * \text{blockIdx.x}$. Each thread first shifts its coordinate by $(-R, -R)$ to the top-left in order to load all necessary data, including padding, into shared memory.

Before performing the computation, the row and col coordinates need to be restored to their original (pre-shift) positions. It's important to note that **the data in shared memory already includes the (-R, -R) offset**, since it was loaded after the coordinate shift.

This means that when a thread with $(\mathrm{threadIdx.x},\ \mathrm{threadIdx.y}) = (0, 0)$ loads data, the corresponding data must be multiplied with the kernel element $F(0, 0)$. Similarly, when $(\mathrm{threadIdx.x},\ \mathrm{threadIdx.y}) = (2R, 2R)$, the data is multiplied with $F(2R, 2R)$.

Based on this, we can conclude that in our implementation, we need two loops ranging from $[0, 2R + 1)$ to perform the convolution calculation.