Programming Massively Parallel Processors

Chapter07 Convolution

1944023 JiYeong Yi

Chapter07

Convolution

Table of contents

7.1 Background

- Convolution is an array operation in which each output element is a weighted sum of the corresponding input element and a collection of input elements that are centered on it.

- The weights that are used in the weighted sum calculation are defined by a filter array, commonly referred to as the convolution kernel (convolution filter).

- 1D convolution is mathematically defined as a function that takes an input data array of n elements $[x_0, x_1, \ldots, x_{n-1}]$ and a filter array of $(2r + 1)$ elements $[f_0, f_1, \ldots, f_{2r}]$ and returns an output data array y:

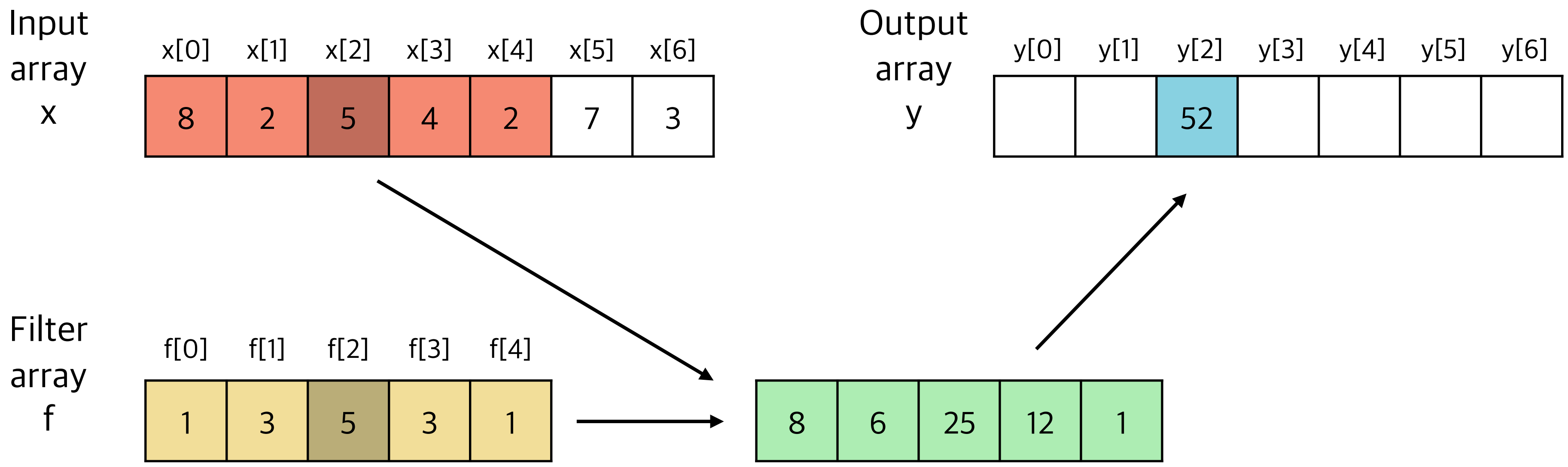
$$y_i = \sum_{j=-r}^{r} f_{r+j} \times x_{i+j}$$

7.1 Background

1D convolution (calculation for y[2])

$$y_i = \sum_{j=-r}^{r} f_{r+j} \times x_{i+j}$$

- y[2] = f[0]*x[0] + f[1]*x[1] + f[2]*x[2] + f[3]*x[3] + f[4]*x[4]
  = 1*8 + 3*2 + 5*5 + 3*4 + 1*1 = 52



7.1 Background

- In a 2D convolution the filter f is also a 2D array.

- If we assume that the dimension of the filter is $(2r_x+1)$ in the x dimension and $(2r_y+1)$ in the y dimension, the calculation of each P element can be expressed as follows:

$$y_i = \sum_{j=-r_y}^{r_y} \sum_{k=-r_x}^{r_x} f_{r_y+j, r_x+k} \times N y_{+j,\ x+k}$$

7.1 Background

$$y_i = \sum_{j=-r_y}^{r_y} \sum_{k=-r_x}^{r_x} f_{r_y+j, r_x+k} \times N y_{+j,\ x+k}$$

2D convolution (calculation for P[2][2])

Input array N

1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	5	6
5	6	7	8	5	6	7
6	7	8	9	0	1	2
7	8	9	0	1	2	3

Output array P

		321				

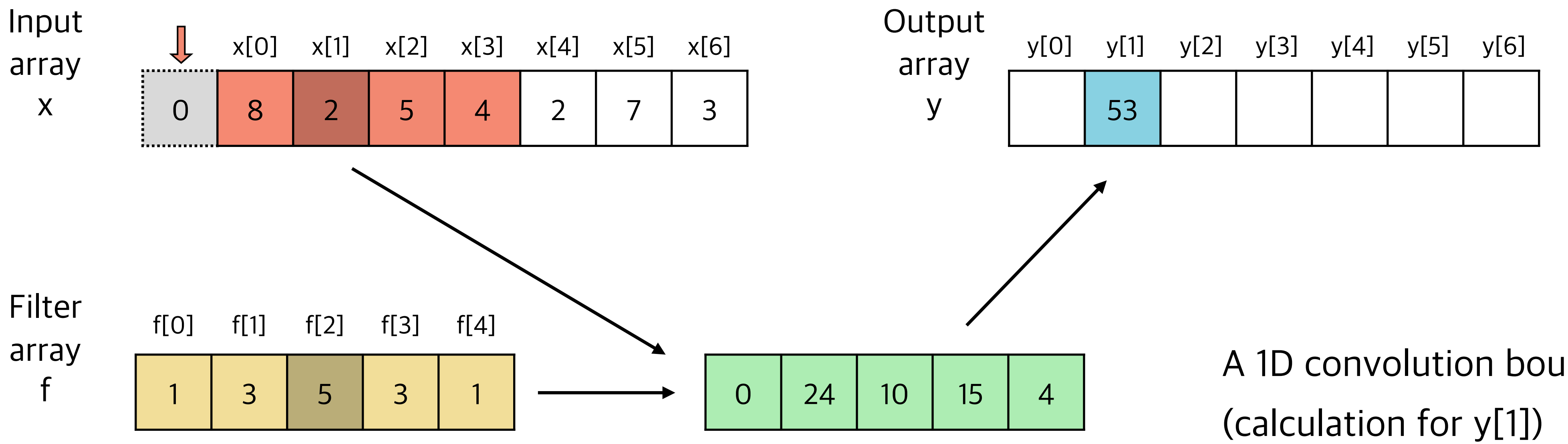
Filter array f

1	2	3	2	1
2	3	4	3	2
3	4	5	4	3
2	3	4	3	2
1	2	3	2	1

1	4	9	8	5
4	9	16	15	12
9	16	25	24	21
8	15	24	21	16
5	12	21	16	5

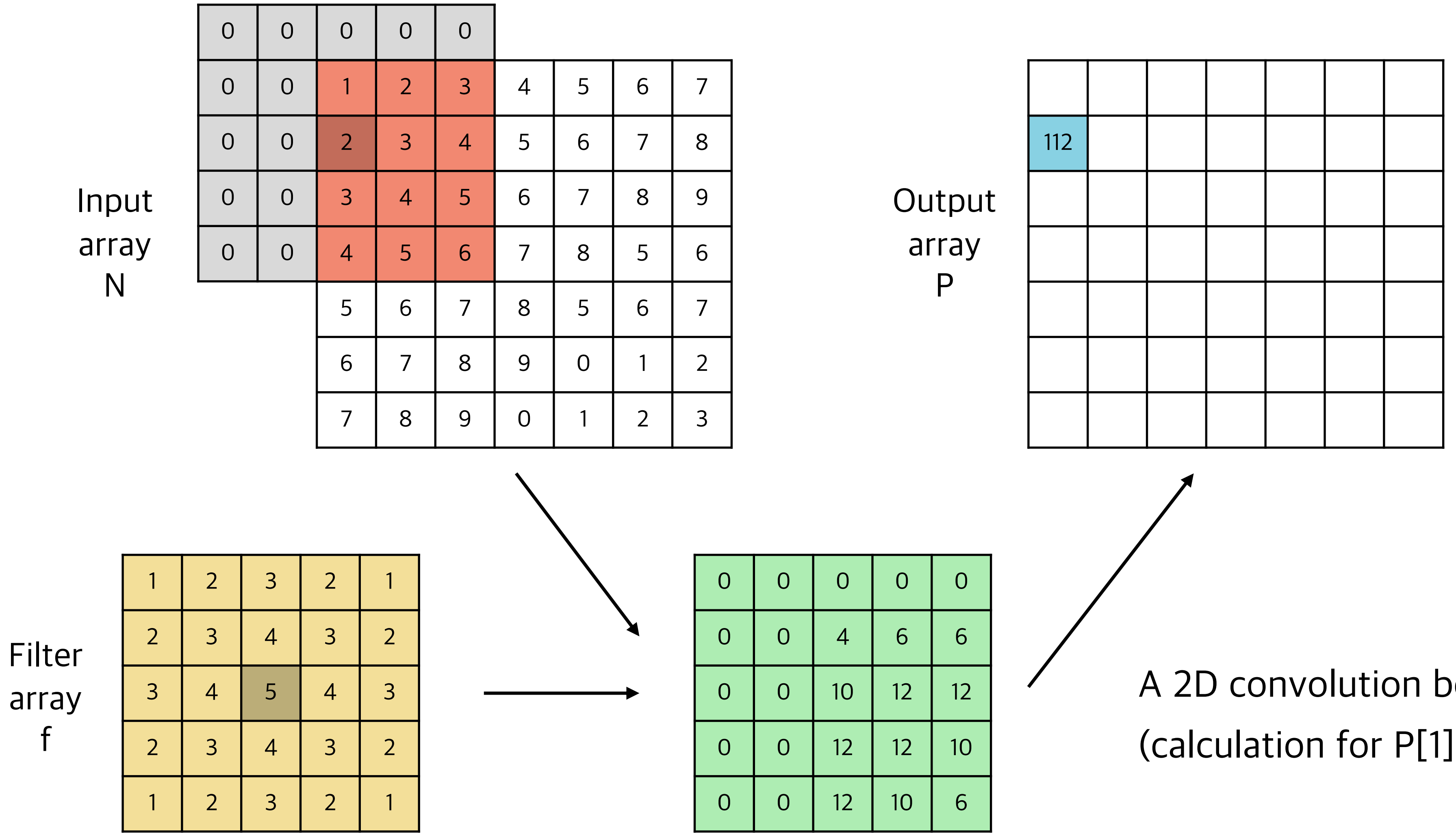
7.1 Background

- Boundary conditions naturally arise in computing output elements that are close to the ends of an array.
- A typical approach to handling such boundary conditions is to assign a default value to theses missing x elements. These missing elements are typically referred to as **ghost cells** in the literature.



A 1D convolution boundary condition.
(calculation for y[1])

7.1 Background

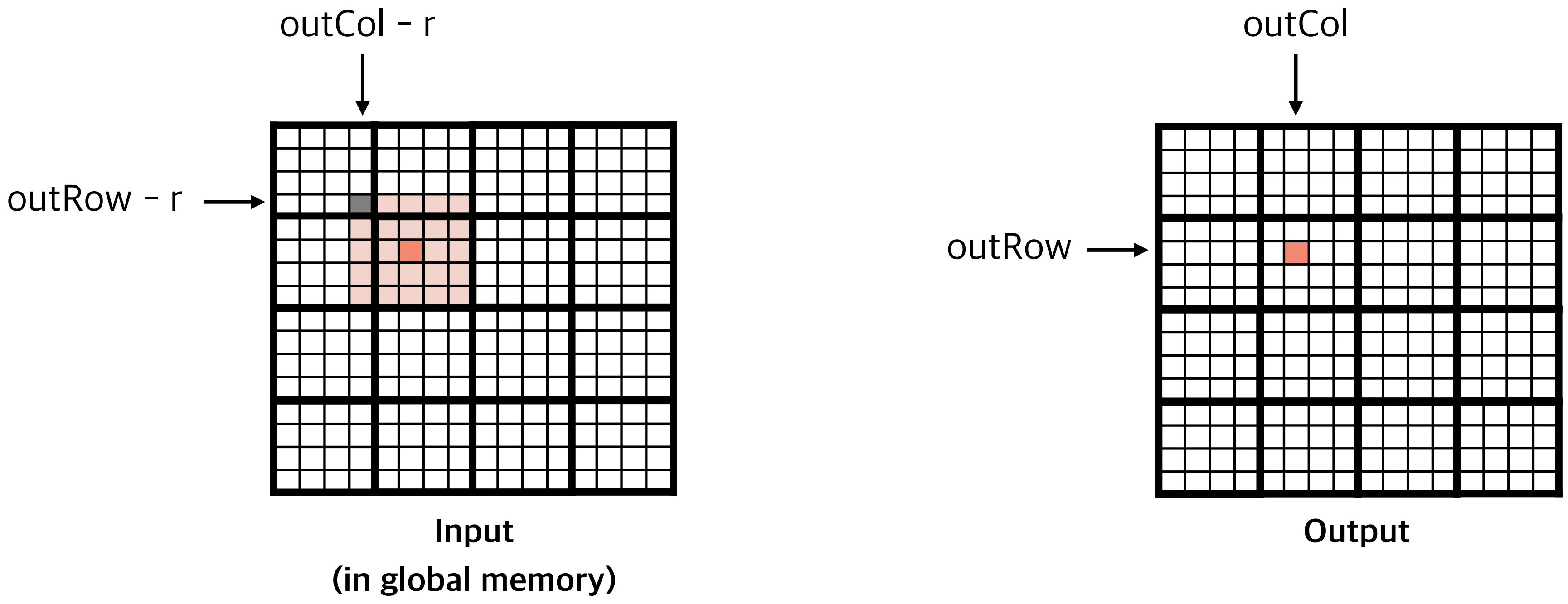


A 2D convolution boundary condition.
(calculation for P[1][0])

7.2 Parallel convolution: a basic algorithm

```
Pvalue += F[fRow][fCol] * N[inRow*width + inCol];
```

- Every thread is assigned to calculate an output pixel whose x and y indices are the same as the thread's x and y indices.
- A serious problem is memory bandwidth. The ratio of floating-point arithmetic calculation to global memory accesses is only about 0.25 OP/B (2 operation for every 8 bytes loaded).



7.3 Constant memory and caching

- Three properties of Filter array F
  1. The size of F is typically small; the radius of most convolution filters is 7 or smaller.
  2. The contents of F do not change throughout the execution of the convolution kernel.
  3. All threads access the filter elements.

  → These three properties make the filter an excellent candidate for **constant memory and caching**.

- Constant memory
  1. Constant memory is quite small, currently at 64 KB.
  2. The value of constant memory variable cannot be modified by threads during kernel execution.
  3. Like global memory variables, constant memory variables are visible to all thread blocks.

7.3 Constant memory and caching

- To declare an F array in constant memory, the host code declares it a global variable as follows:

```
#define FILTER_RADIUS 2
__constant__ float F[2*FILTER_RADIUS + 1][2*FILTER_RADIUS + 1];
```

- The contents of the F_h can be transferred from the host memory to F in the device constant memory as follows:

```
//cudaMemcpyToSymbol(dest, src, size)
cudaMemcpyToSymbol(F, F_h, (2*FILTER_RADIUS + 1)*(2*FILTER_RADIUS + 1)*sizeof(float));
```

7.3 Constant memory and caching

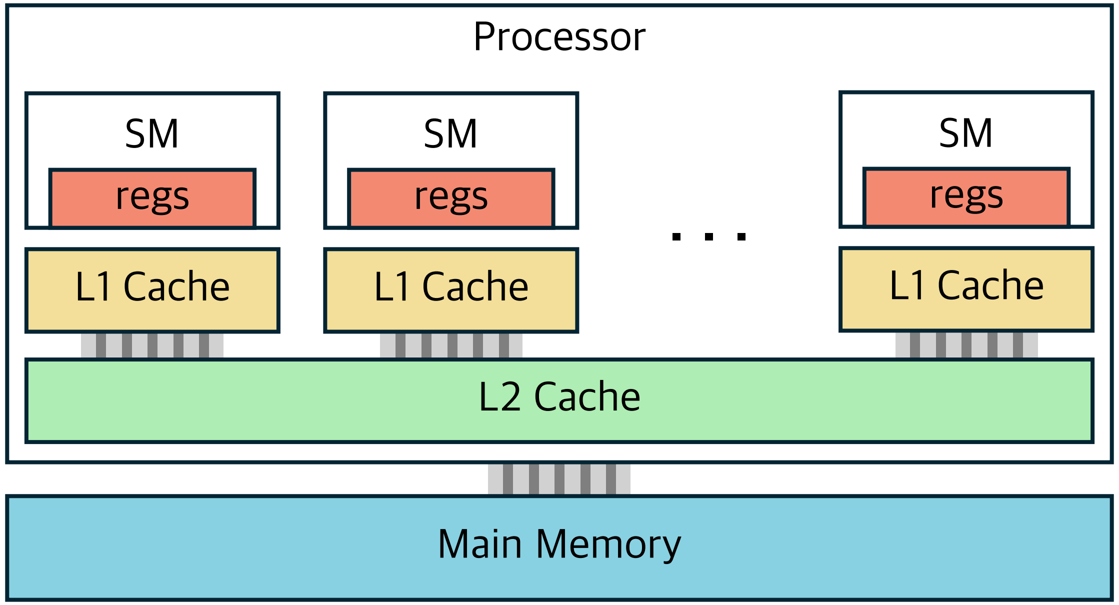
- To mitigate the effect of memory bottleneck, modern processors commonly employ **on-chip cache memories**, or **caches**, to reduce the number of variables that need to be accessed from the main memory (DRAM).

- Caches are “**transparent**” to programs. The processor hardware will automatically retain the most recently or frequently used variables in the cache and remember their original global memory address.

- Modern processors often employ **multiple levels of caches**.
  The numbering convention for these cache levels reflects the distance to the processor.

7.3 Constant memory and caching

- L1 cache is the cache that is directly attached to a processor core. It runs at a speed close to that of the processor in both latency and bandwidth. However, an L1 cache is small, typically between 16 and 64 KB in capacity.
- L2 cache is larger, in the range of a few hundred kilobytes to a small number of MBs but can take tens of cycles to access. L2 cache is typically shared among streaming multiprocessors (SMs) in a CUDA device.



7.3 Constant memory and caching

- 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.

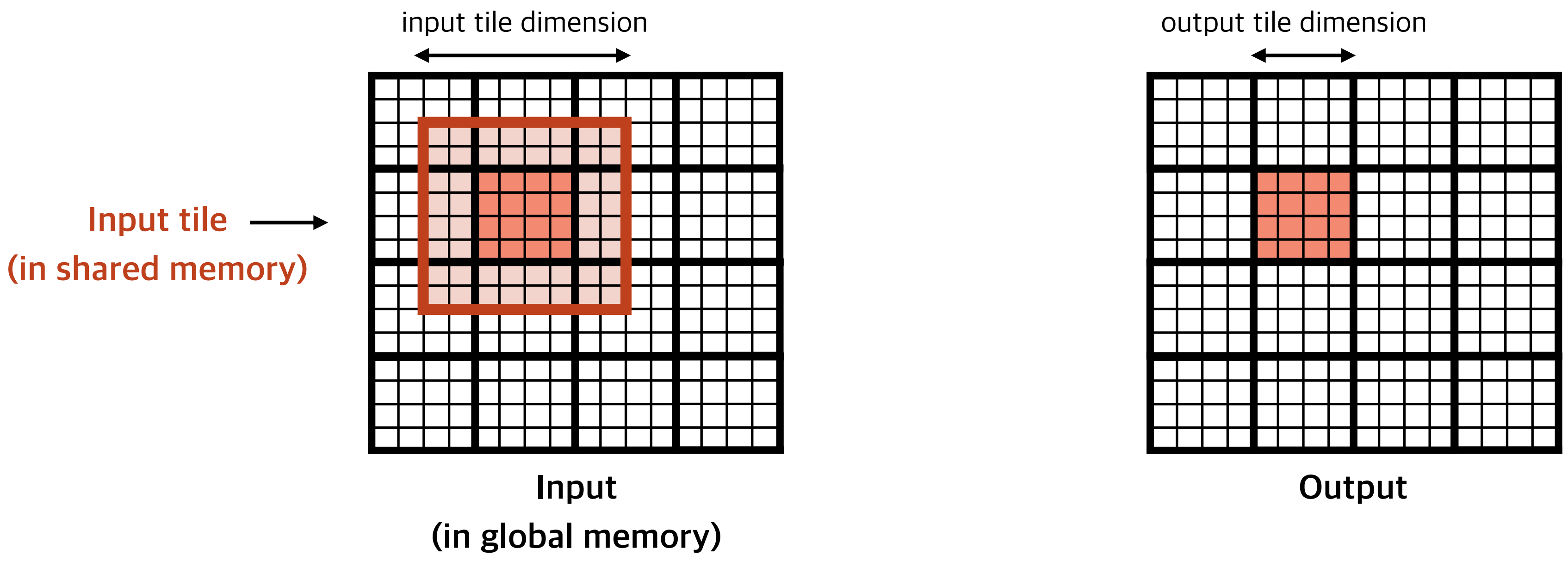
- With the utilization of constant memory and caching, the ratio of floating-point arithmetic calculations to global memory accesses is approximately doubled, reaching about 0.5 OP/B (2 operations for every 4 bytes loaded).

```
Pvalue += F[fRow][fCol] * N[inRow*width + inCol];
```

constant memory
access

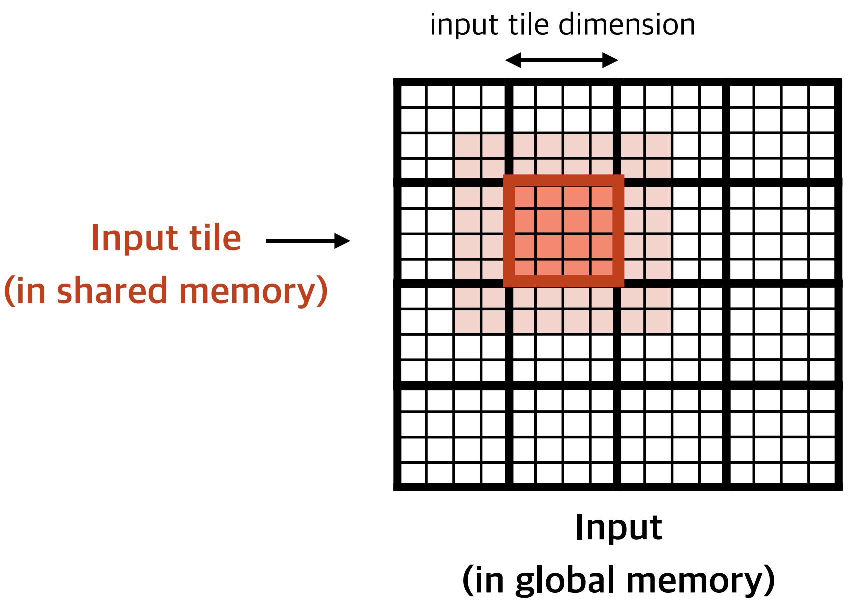
7.4 Tiled convolution with halo cells

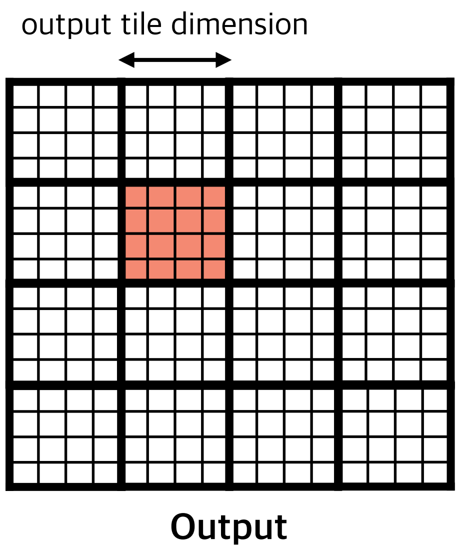
- In a tiled algorithm, threads collaborate to load input elements into an on-chip memory for subsequent use of these elements. The convolution input tiles are larger than the output tiles.
- Launching thread blocks whose dimension matches that of the input tile size simplifies the loading of the input tiles, as each thread needs to load just one input element. However, some of the threads need to be disabled during the calculation of output elements, which can reduce the efficiency of execution resource utilization.



7.5 Tiled convolution using caches for halo cells

- There is a significant probability that by the time a block needs its halo cells, they are already in L2 cache because of the accesses by its neighboring blocks.
- Tiled convolution algorithm using caches uses the same dimension for input and output tiles and loads only the internal elements of each tile into the shared memory.





7.6 Summary

- Methodology

  1. Comparing the performance of 2D convolution algorithms on CPU and GPU.

  2. The operation was repeated for 10 times, and the average values were compared.

  3. Matrix size: 1000x1000, 2000x 2000, 3000x 3000
     Filter array size: 5x5
     GPU block size: 32x32

Ubuntu Version	Ubuntu 22.04.2 LTS
CUDA Version	12.1

NVIDIA GeForce RTX 3080	
SM Count	68
Max resident threads per SM	1536
Max number of resident blocks per SM	16
Threads in warp	32
Max threads per block	1024
Max thread dimensions	(1024, 1024, 64)
Max grid dimensions	(2^{31}–1, 2^{16}–1, 2^{16}–1)
Shared Mem per SM	48 KB
Registers per SM	64 KB
Total constant Mem	64 KB
Total global Mem	10 GB

7.6 Summary

- Results

	CPU Execution Time	GPU Execution Time	GPU (constant Filter) Execution Time	GPU (Tile: 32) Execution Time
1000 x 1000	60.938818 (ms)	55.021100 (us)	44.586900 (us)	47.881500 (us)
2000 x 2000	0.243245 (s)	0.188105 (ms)	0.175614 (ms)	0.172437 (ms)
3000 x 3000	0.510032 (s)	0.395126 (ms)	0.381010 (ms)	0.290755 (ms)