Practice 2: 1D Stencil

Objective: To understand how to implement a basic parallel computation pattern called “Stencil”.

Stencil:

- A computation pattern in which threads read inputs from a fixed neighborhood in an array (data will be repeatedly accessed).
- Examples of stencil in more than one dimension are 2D Von Neumann, 2D Moore, and 3D Von Neumann.
- Stencil is commonly founded in scientific simulation.
- We can think of stencil as a *several-to-one* mapping in memory.
- In Stencil, we need to cooperate threads.

Using Per-Block Shared Memory:

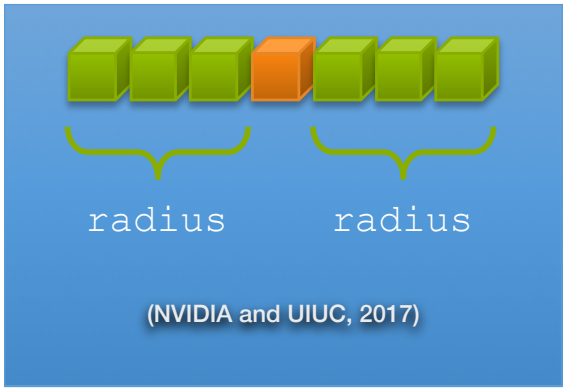
- Communication among threads within a block takes place on *shared memory*.
- In CUDA C, we use keyword “__shared__” to declare a variable on the shared memory.
- The variable declared in this way is invisible to threads in other blocks.
- However, we may need to handle issues like *data race*.

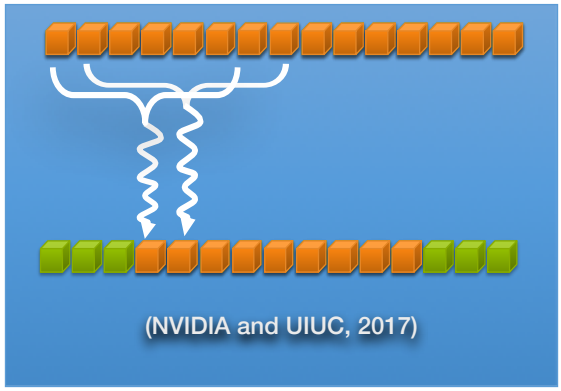
Synchronization Barrier:

- CUDA provides “__syncthread()” a function call to mark a synchronization barrier,
- Threads within each block waits at the marking point until all threads in the block have reached it, and then they are all released to execute the next instruction.
- Synchronization barriers are used to prevent data racing (RAW, WAR, WAW).

1D Stencil:

- Consider applying 1D stencil pattern to a 1D array, i.e., each output element is the sum of input elements within a radius. If the radius is 3, then each output element is the sum of 7 input elements.





Practice 2.1: Implement a C program to compute 1D stencil within radius $k = 3$.

Practice 2.2: Implement a CUDA C program to compute 1D stencil within radius $k = 3$ by using just global device memory.

Practice 2.3, : Implement a CUDA C program to compute 1D stencil within radius $k = 3$ by using per-block shared memory.