I. DIFFUSION EQUATION

Equation:

$$\frac{\partial \phi}{\partial t} = D\nabla^2\phi, \tag{1}$$

Boundary Condition:

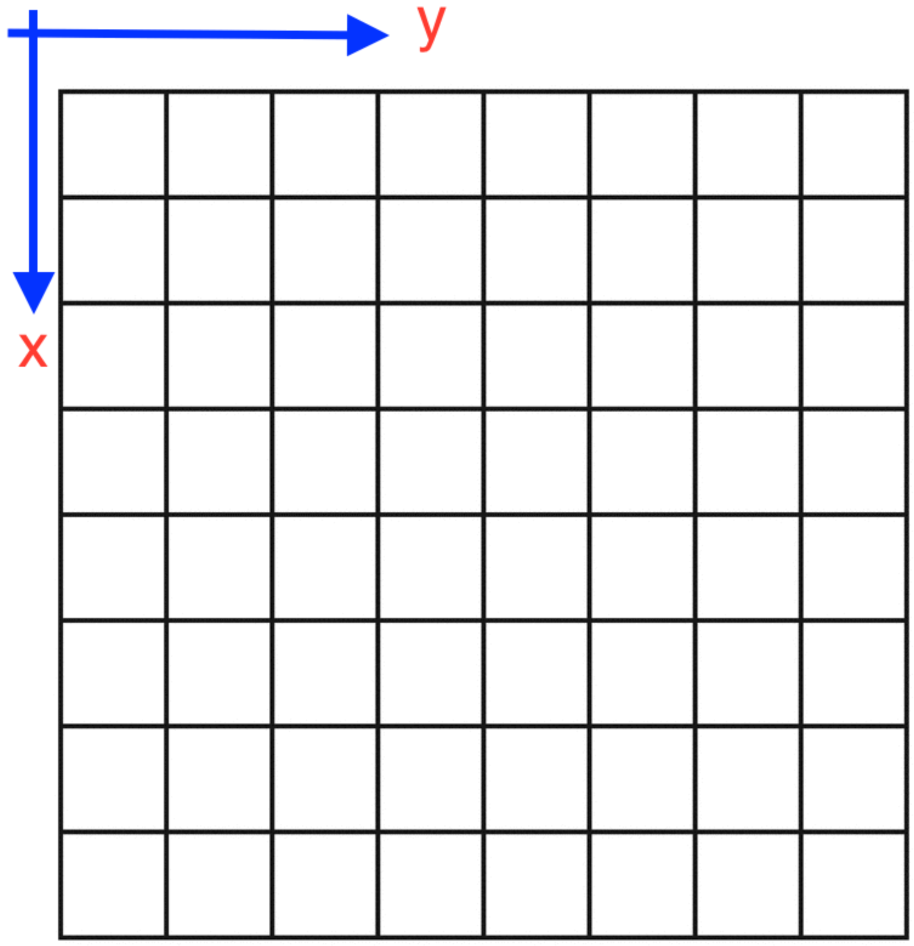
$$\begin{aligned} \phi(x=0) = \phi(x=L) &= 0 \\ \phi(y=0) &= 0 \\ \phi(y=L) = \sin(\frac{\pi}{L}x)\sinh(\pi) \end{aligned} \tag{2}$$

Analytical solution for the steady-state:

$$\phi(x,y) = \sinh(\frac{\pi}{L}y)\sin(\frac{\pi}{L}x) \tag{3}$$

Finite different update rule:

$$\nabla^2\phi \approx \frac{\sum_{i=1}^{8}\phi(\vec{r}+\Delta\vec{r}_i) - 8\phi(\vec{r})}{3\Delta x^2} \tag{4}$$



Copy the whole "Old" array into the global memory.

Using Multiple Blocks.

Each Block has multiple threads

When updating the "Current" array, each thread directly fetch data from the global memory.

Disadvantage:

1, Fetching data from the global memory is slow.

2, Need to synchronize all the threads. But cuda (version¡9.0) do not provide synchronization mechanism between blocks. Need cudaMemcpy for every iteration step.

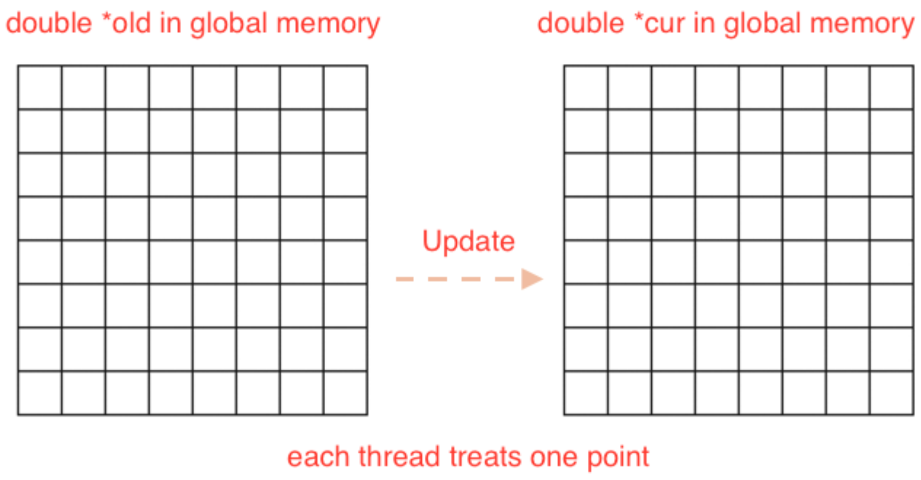

double *old in global memory
double *cur in global memory
Update
each thread treats one point

FIG. 2. Multi-Blocks + Global memory

Copy the whole "Old" array into the global memory.

Using a single block of threads. This is because there is a syncthreads() function which can synchronize all the threads in the block.

In this way, we avoid so many cudaMemcpys. Disadvantage:

1, Fetching data from the global memory is slow.

2, Each thread need to cover multiple points.

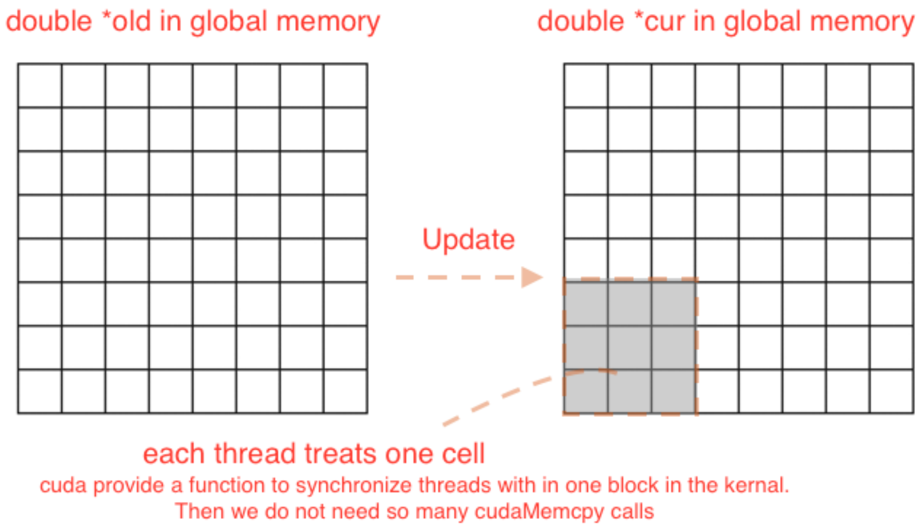

double *old in global memory
double *cur in global memory
Update
each thread treats one cell
cuda provide a function to synchronize threads with in one block in the kernal.
Then we do not need so many cudaMemcpy calls

FIG. 3. Single Block + Global memory.

Copy part of the ”Old” array into the shared memory.

Threads fetch data from shared memory is fast.

Still need to synchronize all the blocks, using cudaMemcpy every iteration step.

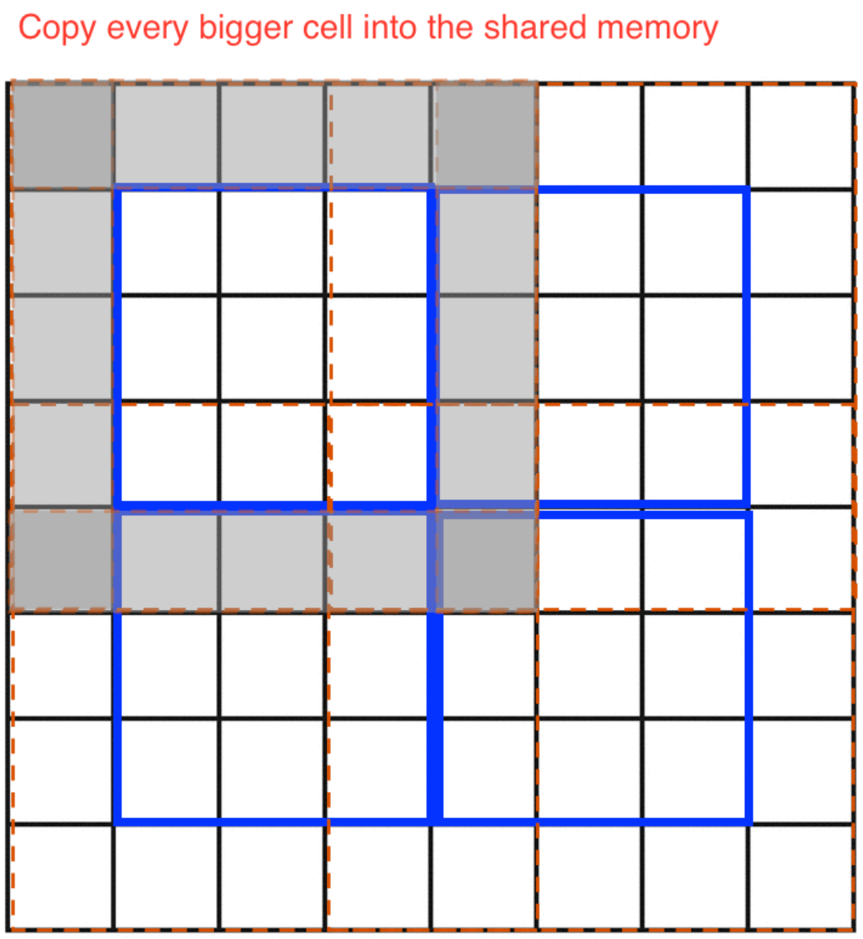


FIG. 4. Single Block + Global memory.