2. The code below carries out a “nearest neighbor” or “stencil” computation. This class of algorithm appears frequently in image processing applications. The memory reference pattern for matrix **b** exhibits reuse in 3 dimensions. Your task is to develop a C/CUDA version of this code that initializes **b** on the host and then uses tiling on the GPU to exploit locality in shared memory across the 3 dimensions of **b**:

```
#define n 64
float a[n][n][n], b[n][n][n];
for (i=1; i<n-1; i++)
   for (j=1; j<n-1; j++)
          for (k=1; k<n-1; k++) {
 a[i][j][k]=0.8*(b[i-1][j][k]+b[i+1][j][k]+b[i][j-1][k]
 + b[i][j+1][k]+b[i][j][k-1]+b[i][j][k+1]);
     }
```

a) Evaluate the performance of computing a tiled versus non-tiled implementation in your GPU application. Explore what happens when you change the value of n. Consider the performance for at least two additional values for n.

n: 64

```
non tiled method: 0.007040 milliseconds
tiled method: 0.002048 milliseconds
```

It can be seen that the runtime using tiling method has a significant decrease compared to the runtime without using tiling method, because tiling uses shared memory and reduces the frquency of device memory accesses, thus reducing the access latency.

n: 128

```
non tiled method: 0.007936 milliseconds
tiled method: 0.002112 milliseconds
```

n: 256

```
non tiled method: 0.009888 milliseconds
tiled method: 0.002144 milliseconds
```

As n increases, the running time of both methods rises, but the increase in the tiling method is relatively much smaller. However, when n is too large, the shared memory out of range situation will occur.

b) Explore and report on other optimizations to accelerate your code on the GPU?

If the number of threads is too small, the program will not have enough parallelism and there will be no other warp for I/O operations. if the number of threads is too large, the number of registers in each SM is limited and the more threads there are, the fewer blocks can be parallelized, which will lead to a slow running program and not reaching the bandwidth bottleneck. Therefore, the number of threads needs to be adjusted appropriately to obtain maximum acceleration during acceleration with cuda.