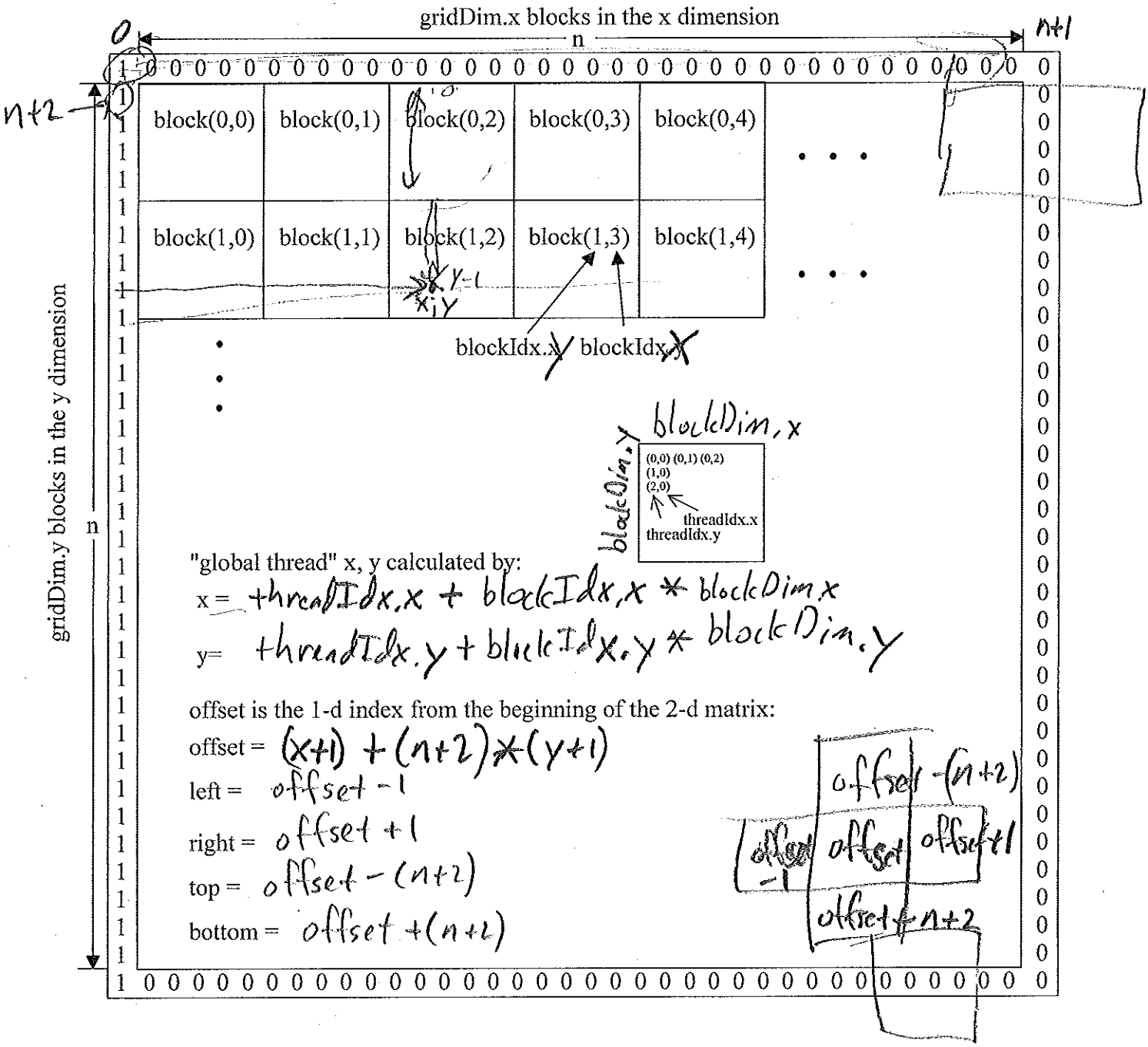
HW #5: Implement the 2D SOR algorithm on cudacube using CUDA on the GPU.

Simplifying assumption: square n x n array (Let's handle n not matching the dimensions of the grid of threads)

2D SOR - on each iteration replace all interior values by the average of their four nearest neighbors



a) Maximum threads per block is 512 (2^9). If we want to make it 2-dimensional (and square), what would the dimensions of the thread block?

#define DIM 16

#define threadsPerBlock DIM x DIM

dim3 thread(DIM , DIM);

dim3 block((n+(DIM-1))/DIM , _____);

<<< block , thread >>>

b) Design the host's algorithm:

- allocate 1d chunk of size (n+2)*(n+2)*sizeof(float)
- initialize to all 0's, except left-hand column. (lets use array "values")
- cudaMalloc same sized dev_current_array on device
- " " " dev_new_array on device
- cudaMemcpy values to dev_current_array
- cudaMalloc dev_deltas array with one float per thread block

do

- run kernel that uses dev_current_array to calculate device_new_array and then have each thread block calculate their max. delta value and put it into dev_deltas
- cudaMemcpy dev_delta values from device and have host determine the "global" max. delta.
- swap the pointers to dev_current_array and dev_new_array

while (delta > threshold)

c) Design the device's kernel: In thread memory declare thread_deltas array

- calculate x, y, offset, left, right, top, bottom
- use these and dev_current_array to calculate average to put in dev_new_array and use to fill thread_deltas spot
- have each block of threads do binary-tree reduction to find max delta
- have thread 0 fill dev_deltas slot with it.