EXTRA CREDIT HW #8 (Discussed as Lecture 27 -- see answers below): Implement the 2D SOR algorithm using CUDA on the Tesla C2070 GPU.

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

Simplifying assumption: square array nxn interior (Let’s handle n not matching the dimensions of the grid of threads)

a) Maximum threads per block is 1024 (2^{10}). If we want to make it 2-dimensional (and square), what would the dimensions of the thread block (DIM by DIM)? 2^{10} = 2^5 * 2^5, so DIM is 32 and each thread block is 32 by 32

b) If we want to “tile” blocks across the n x n interior of the array, what is the dimension of the grid of blocks?
We want ceiling(n/DIM) which we can calculate by (n + DIM -1)/DIM

#define DIM 32
#define threadsPerBlock 32 * 32
dim3 threads( DIM, DIM);
dim3 blocks( (n + DIM -1)/DIM, (n + DIM -1)/DIM );
c) Design the host’s algorithm:

(NOPE: when allocating host arrays we might want to allocate them in “pinned” (i.e., non-pagable) memory to speed cudaMemcpy back from the GPU device. To do this you need to use cudaMemcpyHostAlloc and cudaMemcpyHostFree(val);

```
cudadc( void**) &val, (n+2)*(n+2)*sizeof(float), cudaMemcpyHostAllocDefault);
```

- allocate val array as 1-D chunk of host memory of size (n+2) * (n + 2) * sizeof(float)
- initialize val to all 0’s, except 1’s down column 0
- cudaMalloc same sized dev_current_array in global memory of GPU device
- cudaMalloc same sized dev_new_array in global memory of GPU device
- cudaMemcpy val array to dev_current_array
- cudaMemcpy val array to dev_new_array to get the exterior values mostly
- allocate block_deltas array on host -- one float value per thread block
- cudaMalloc same size dev_block_deltas array in global memory of GPU device
- do {
  - run kernel on GPU that uses dev_current_array to calculate dev_new_array and then have each thread block calculate its max. delta value and put it into dev_block_deltas array
  - cudaMemcpy dev_deltas array from global memory on the GPU device back to block_deltas array on host and then have the host determine the “global” max. delta
  - swap the pointers to dev_current_array and dev_new_array
- } while (delta > threshold);
- cudaMemcpy the dev_current_array back the host’s val array to get the result;

---

d) Design the device’s kernel:

- in the thread block’s shared memory declare a thread_deltas array for each thread in the block to store its delta
- calculate x, y, offset, left, right, top, bottom -- see page 1
- use these indexes into dev_current_array to calculate average to put into dev_new_array and determine its delta which it puts into its thread_deltas array spot
- have each block of threads do a binary-tree reduction to find the block’s max. delta value (like kernelC code of
- have thread 0 of the of each block fill the dev_block_deltas spot