ECE408/CS483/CSE408 Fall 2020

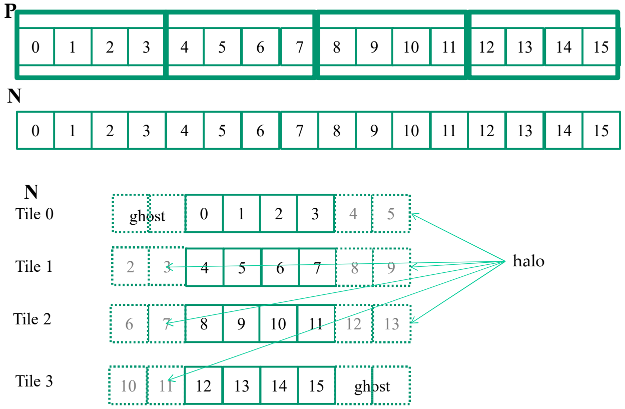
Applied Parallel Programming

Lecture 9: Tiled Convolution

Objective

- To learn about tiled convolution algorithms
  - Some intricate aspects of tiling algorithms
  - Output tiles versus input tiles
  - Three different styles of input tile loading
  - To prepare for MP-4

Tiled 1D Convolution Basic Idea



P

0 1 2 3 | 4 5 6 7 | 8 9 10 11 | 12 13 14 15

N

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

N

Tile 0: ghost 0 1 2 3 4 5

Tile 1: 2 3 4 5 6 7 8 9

Tile 2: 6 7 8 9 10 11 12 13

Tile 3: 10 11 12 13 14 15 ghost

halo

What Shall We Parallelize?

In other words,

What should one thread do?

One answer:

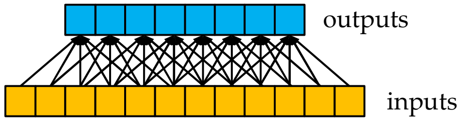
- (same as with vector sum and matrix multiply)
- **compute an output element!**

Should We Use Shared Memory?

In other words,

Can we reuse data read from global memory?

Let's look at the computation again…

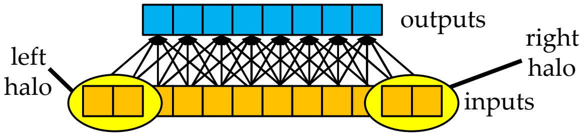


Reuse reduces global memory bandwidth,
so **let's use shared memory**.

What About the Halos?

In other words,

Do we also copy halos into shared memory?



Let's **consider both** possible answers.

How Much Reuse is Possible?

MASK_WIDTH is 5

tile 2 3 4 5 6 7 8 9

- Element 2 is used by thread 4 (1×)
- Element 3 is used by threads 4, 5 (2×)
- Element 4 is used by threads 4, 5, 6 (3×)
- Element 5 is used by threads 4, 5, 6, 7 (4×)
- Element 6 is used by threads 4, 5, 6, 7 (4×)
- Element 7 is used by threads 5, 6, 7 (3×)
- Element 8 is used by threads 6, 7 (2×)
- Element 9 is used by thread 7 (1×)

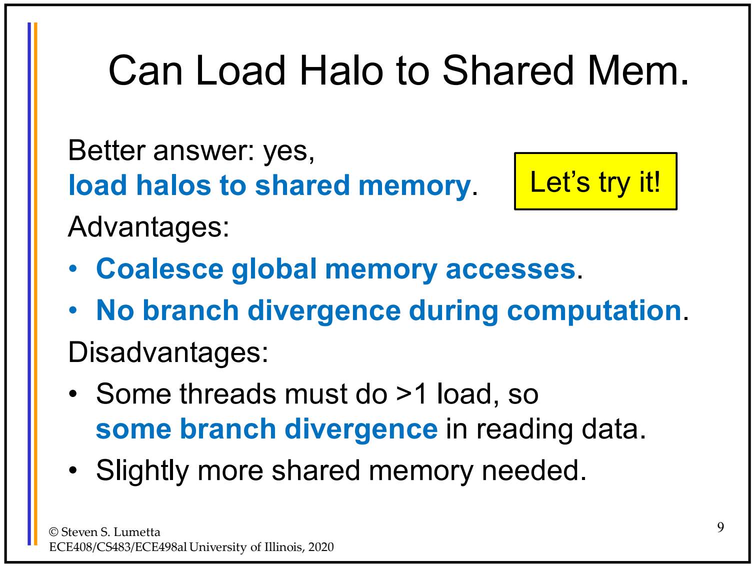
Can Access Halo from Global Mem.

bad idea

One answer: no,
- threads **read halo values**
- directly **from global memory**.

Advantage:
- optimize reuse of shared memory
- (halo reuse is smaller).

Disadvantages:
- **Branch divergence**! (shared vs. global reads)
- Halo **too narrow to fill** a memory **burst**



Can Load Halo to Shared Mem.

Better answer: yes,
load halos to shared memory.

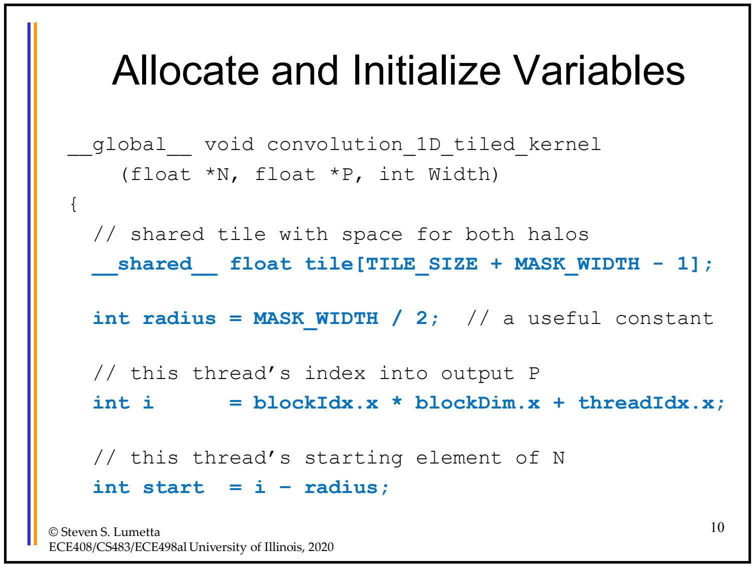
Let’s try it!

Advantages:

- **Coalesce global memory accesses**.
- **No branch divergence during computation**.

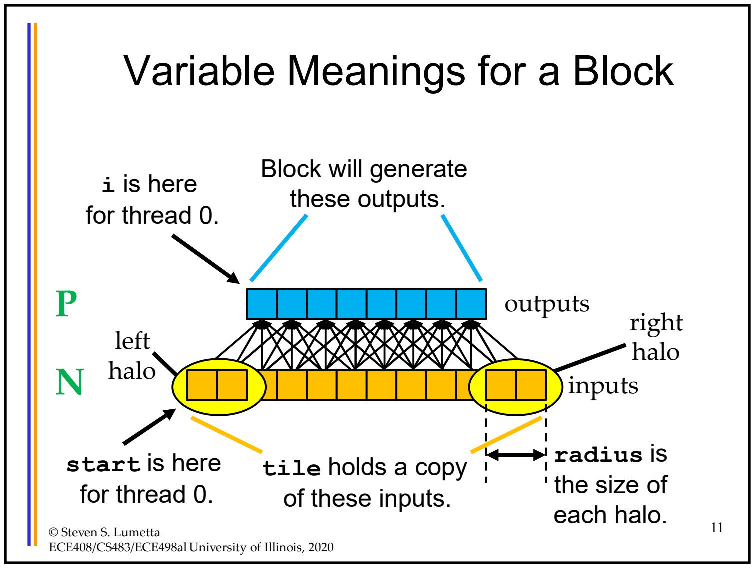
Disadvantages:

- Some threads must do >1 load, so **some branch divergence** in reading data.
- Slightly more shared memory needed.

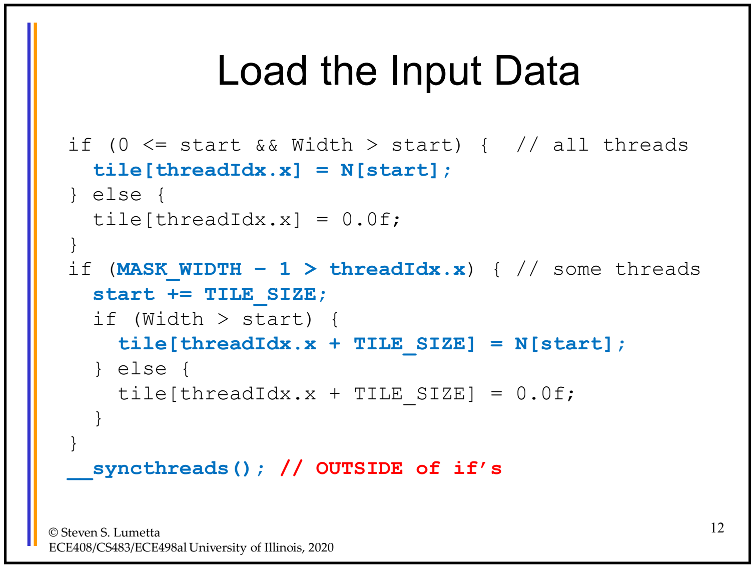


Allocate and Initialize Variables

```
__global__ void convolution_1D_tiled_kernel
    (float *N, float *P, int Width)
{
  // shared tile with space for both halos
  __shared__ float tile[TILE_SIZE + MASK_WIDTH - 1];

  int radius = MASK_WIDTH / 2;  // a useful constant

  // this thread’s index into output P
  int i      = blockIdx.x * blockDim.x + threadIdx.x;

  // this thread’s starting element of N
  int start  = i – radius;
```

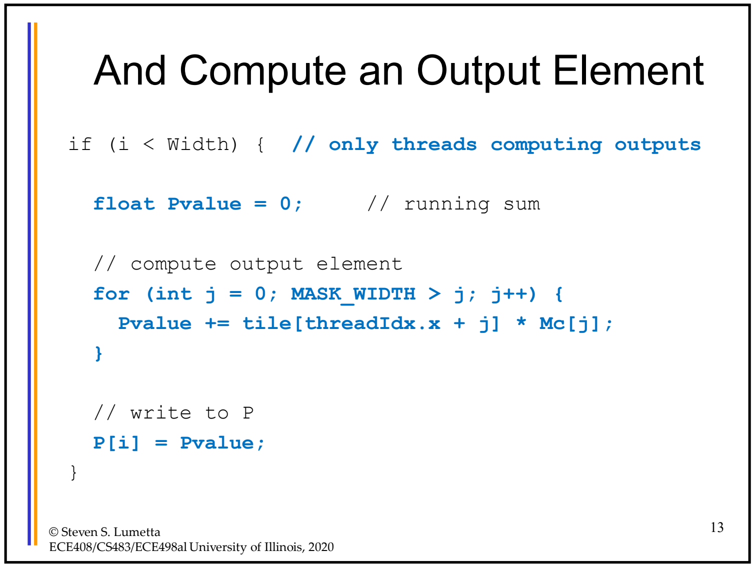


Variable Meanings for a Block



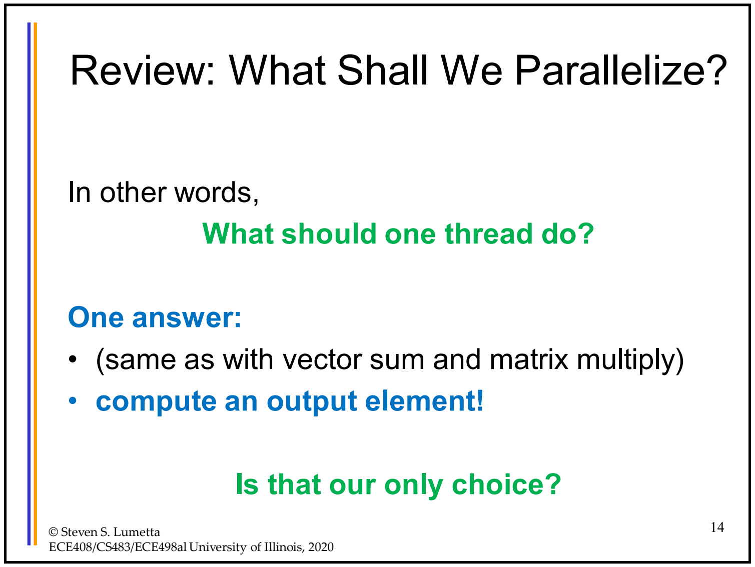
Load the Input Data

```
if (0 <= start && Width > start) {  // all threads
  tile[threadIdx.x] = N[start];
} else {
  tile[threadIdx.x] = 0.0f;
}
if (MASK_WIDTH – 1 > threadIdx.x) { // some threads
  start += TILE_SIZE;
  if (Width > start) {
    tile[threadIdx.x + TILE_SIZE] = N[start];
  } else {
    tile[threadIdx.x + TILE_SIZE] = 0.0f;
  }
}
__syncthreads(); // OUTSIDE of if’s
```



And Compute an Output Element

```
if (i < Width) {  // only threads computing outputs

  float Pvalue = 0;      // running sum

  // compute output element
  for (int j = 0; MASK_WIDTH > j; j++) {
    Pvalue += tile[threadIdx.x + j] * Mc[j];
  }

  // write to P
  P[i] = Pvalue;
}
```



Review: What Shall We Parallelize?

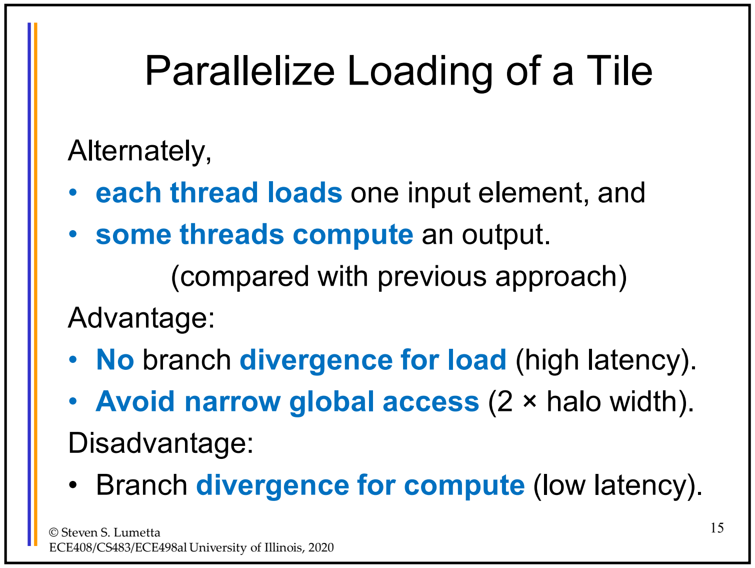
In other words,

What should one thread do?

One answer:

- (same as with vector sum and matrix multiply)
- **compute an output element!**

Is that our only choice?



Parallelize Loading of a Tile

Alternately,

- **each thread loads** one input element, and
- **some threads compute** an output.

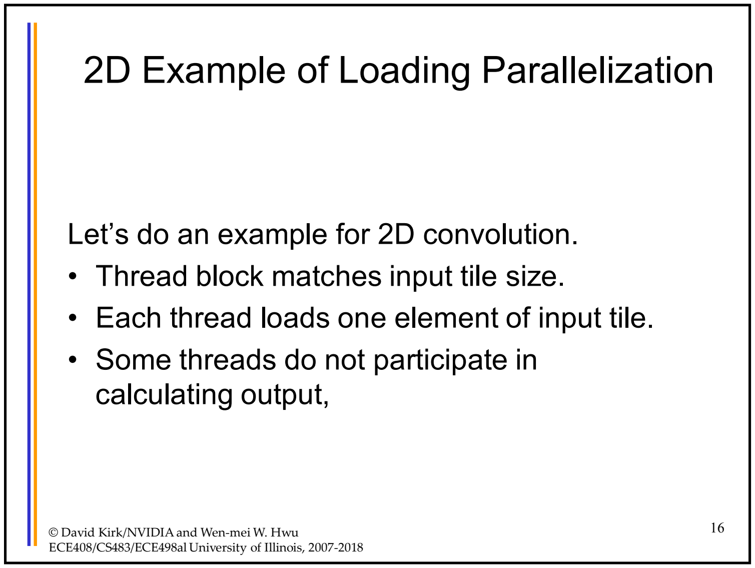
(compared with previous approach)

Advantage:

- **No** branch **divergence for load** (high latency).
- **Avoid narrow global access** (2 × halo width).

Disadvantage:

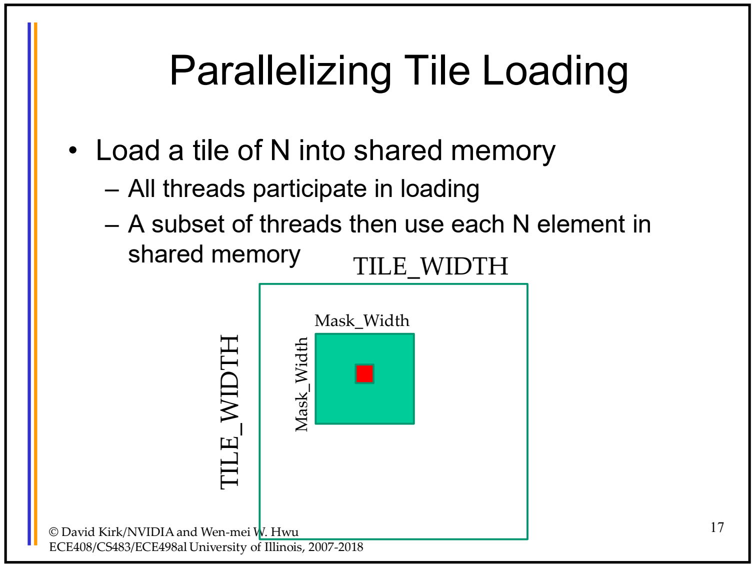
- Branch **divergence for compute** (low latency).



2D Example of Loading Parallelization

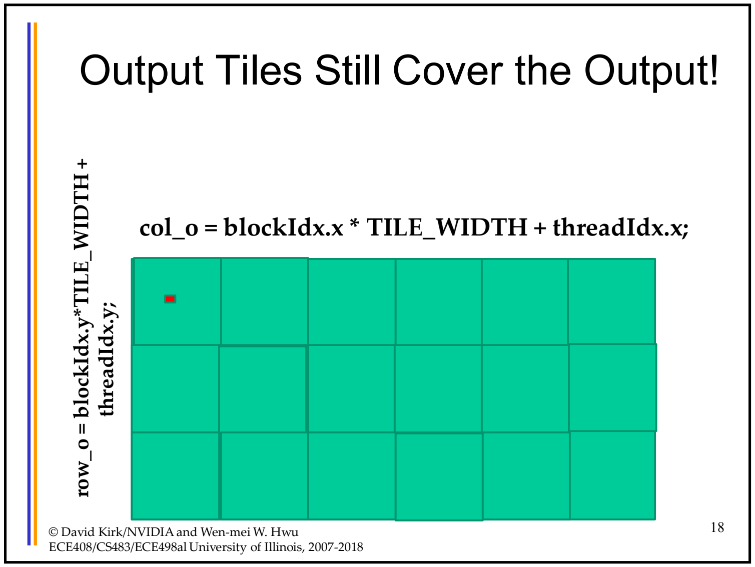
Let’s do an example for 2D convolution.

- Thread block matches input tile size.
- Each thread loads one element of input tile.
- Some threads do not participate in calculating output,



Parallelizing Tile Loading

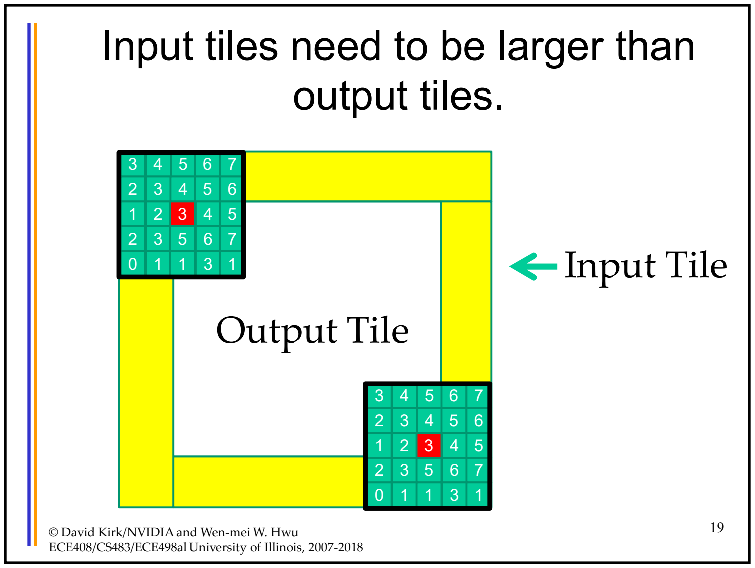
- Load a tile of N into shared memory
  - All threads participate in loading
  - A subset of threads then use each N element in shared memory



Output Tiles Still Cover the Output!

col_o = blockIdx.x * TILE_WIDTH + threadIdx.x;

row_o = blockIdx.y*TILE_WIDTH + threadIdx.y;



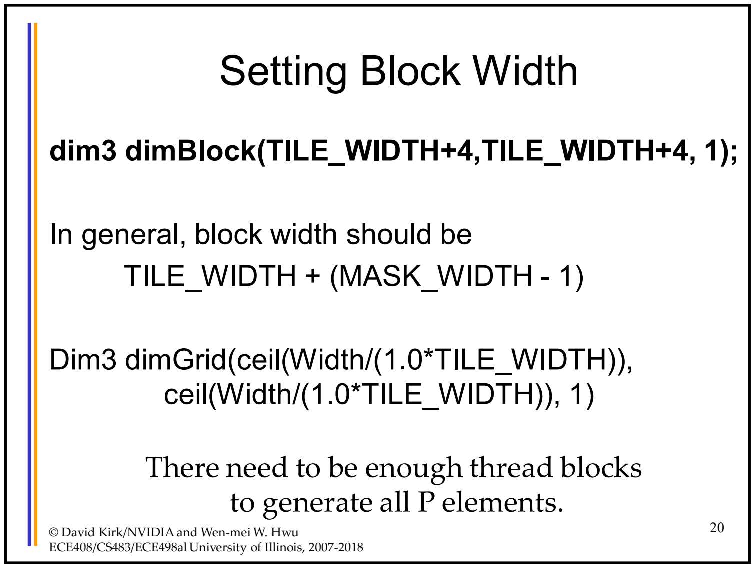
Input tiles need to be larger than output tiles.

3	4	5	6	7
2	3	4	5	6
1	2	3	4	5
2	3	5	6	7
0	1	1	3	1

Input Tile

Output Tile

3	4	5	6	7
2	3	4	5	6
1	2	3	4	5
2	3	5	6	7
0	1	1	3	1



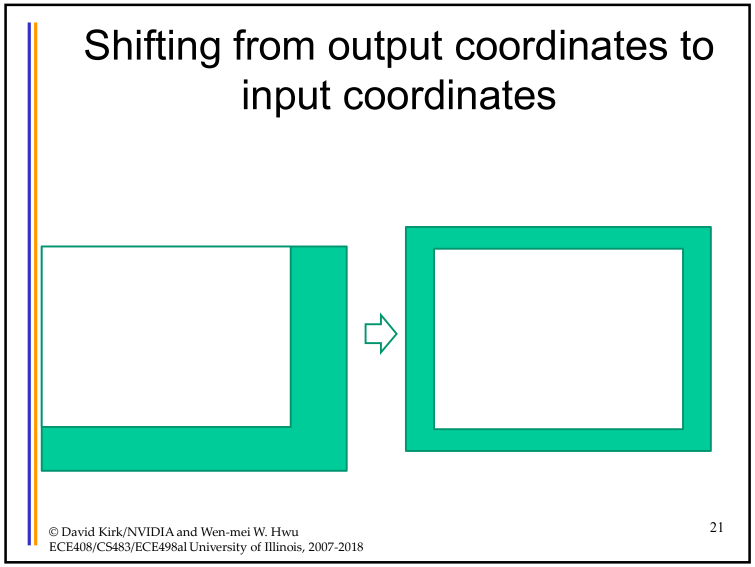
Setting Block Width

dim3 dimBlock(TILE_WIDTH+4,TILE_WIDTH+4, 1);

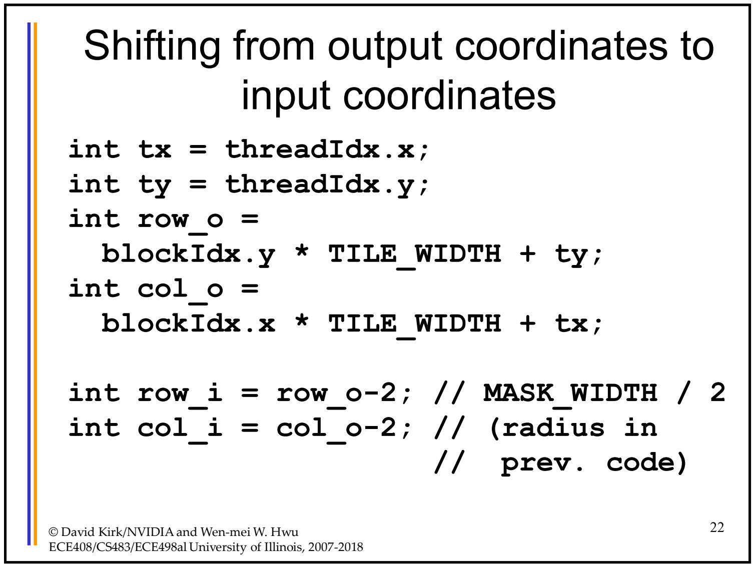
In general, block width should be
TILE_WIDTH + (MASK_WIDTH - 1)

Dim3 dimGrid(ceil(Width/(1.0*TILE_WIDTH)),
ceil(Width/(1.0*TILE_WIDTH)), 1)

There need to be enough thread blocks
to generate all P elements.

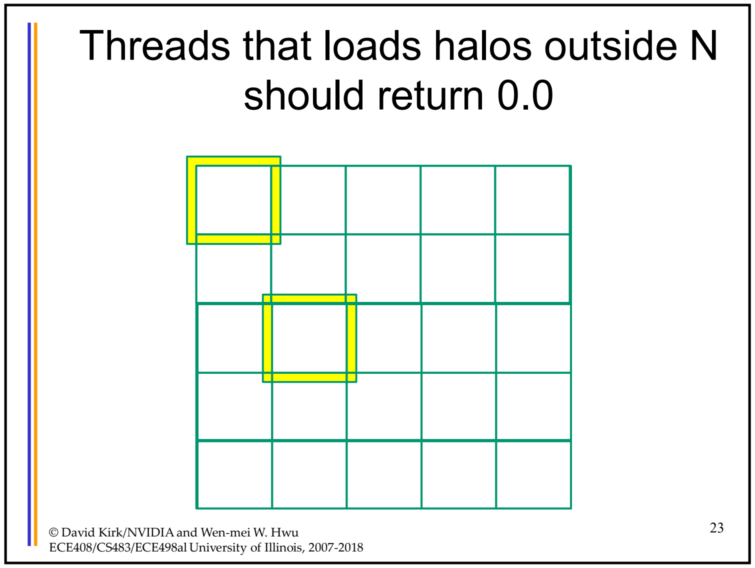


Shifting from output coordinates to input coordinates

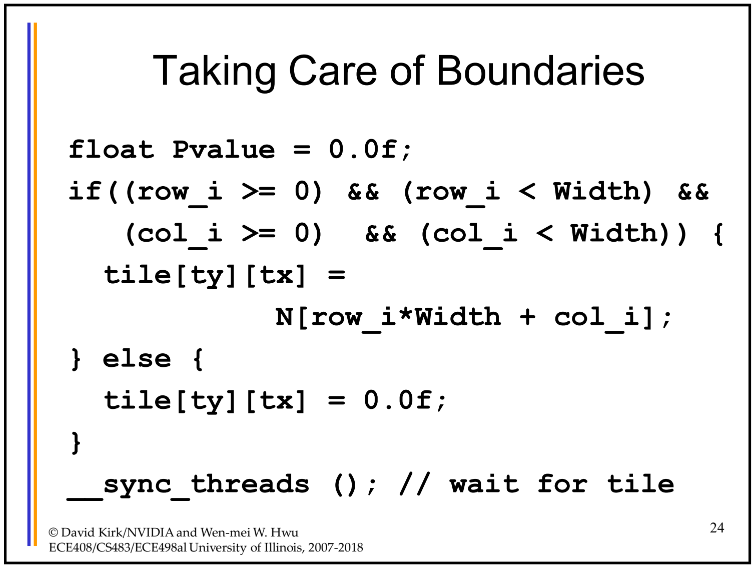


Shifting from output coordinates to input coordinates

```
int tx = threadIdx.x;
int ty = threadIdx.y;
int row_o =
  blockIdx.y * TILE_WIDTH + ty;
int col_o =
  blockIdx.x * TILE_WIDTH + tx;

int row_i = row_o-2; // MASK_WIDTH / 2
int col_i = col_o-2; // (radius in
                     //  prev. code)
```

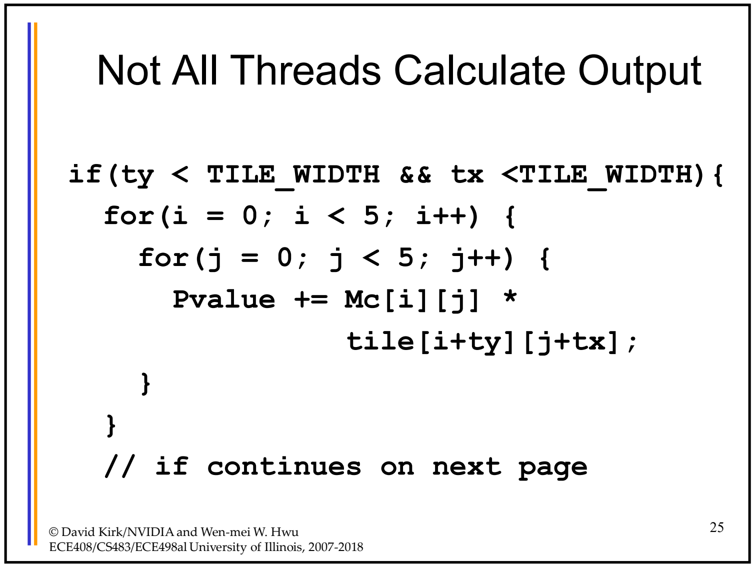


Threads that loads halos outside N should return 0.0



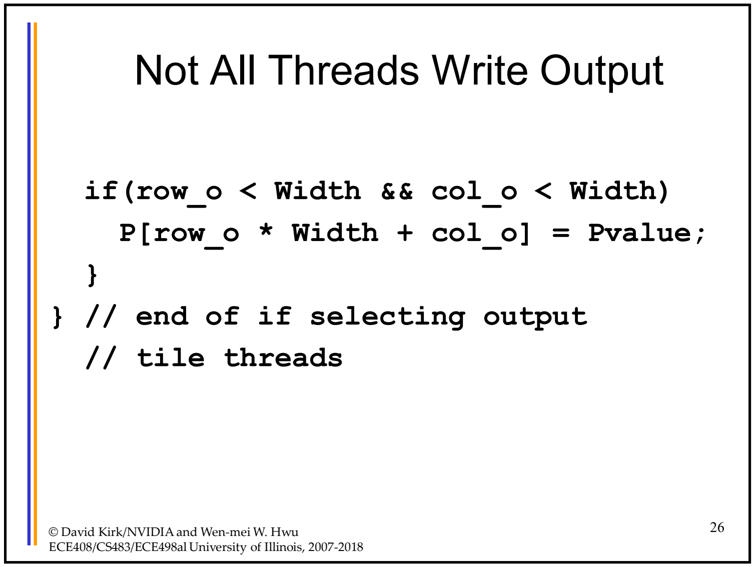
Taking Care of Boundaries

```
float Pvalue = 0.0f;
if((row_i >= 0) && (row_i < Width) &&
   (col_i >= 0)  && (col_i < Width)) {
  tile[ty][tx] =
            N[row_i*Width + col_i];
} else {
  tile[ty][tx] = 0.0f;
}
__sync_threads (); // wait for tile
```



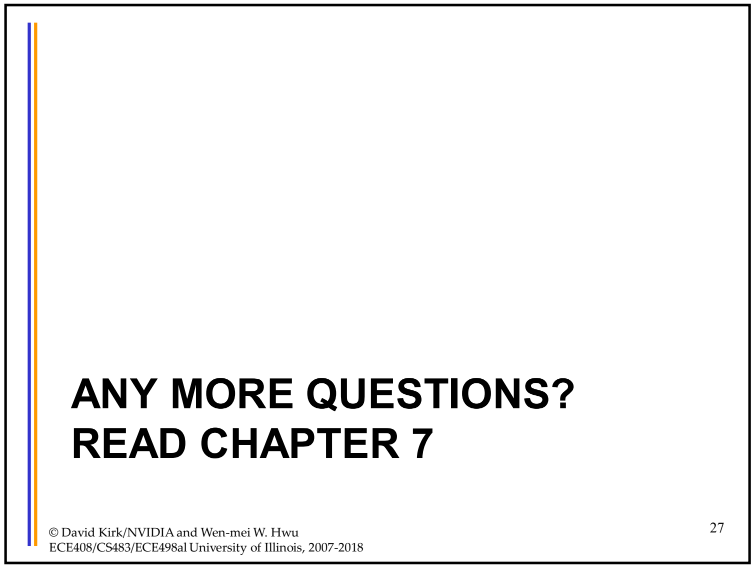
Not All Threads Calculate Output

```
if(ty < TILE_WIDTH && tx <TILE_WIDTH){
  for(i = 0; i < 5; i++) {
    for(j = 0; j < 5; j++) {
      Pvalue += Mc[i][j] *
                tile[i+ty][j+tx];
    }
  }
  // if continues on next page
```



Not All Threads Write Output

```
    if(row_o < Width && col_o < Width)
      P[row_o * Width + col_o] = Pvalue;
    }
  } // end of if selecting output
    // tile threads
```



**ANY MORE QUESTIONS?
READ CHAPTER 7**