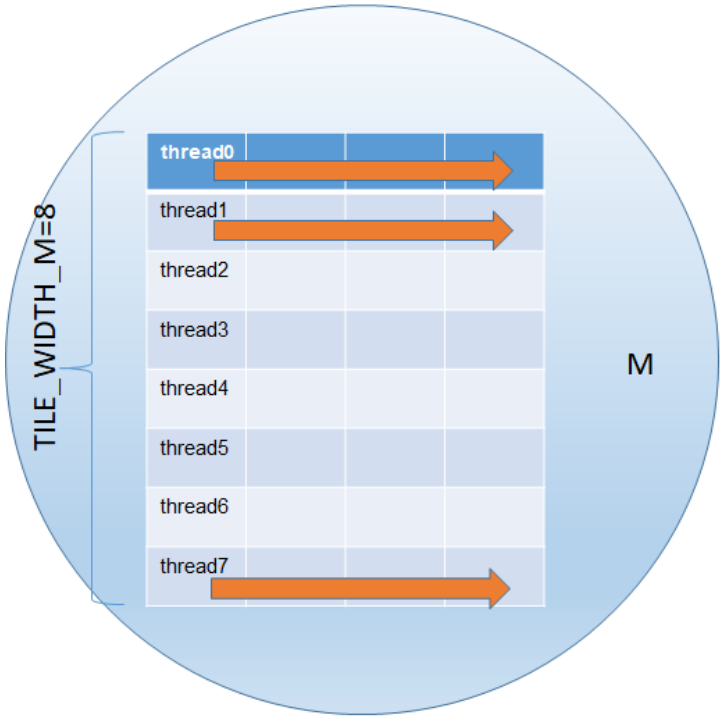
Assignment 2

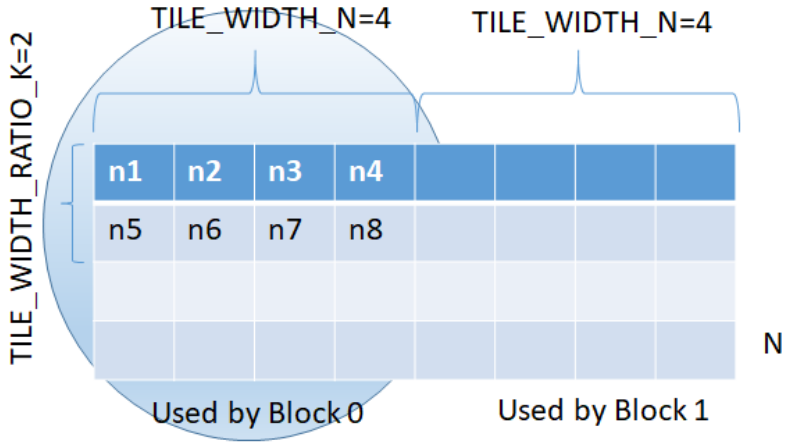
CSD2170/CS300 Programming Massively Parallel Processors

William Zheng

Assignment 2 Example

- Use the example illustrated in the CSD2170A2.PDF.





In this example, assume TILE_WIDTH_N=4, TILE_WIDTH_M=8. m = 8, k= 4, n = 8.

Block size for kernel launch = (TILE_WIDTH_M,1,1).
# of blocks in x dimension = (m+TILE_WIDTH_M-1)/TILE_WIDTH_M=1
# of blocks in y dimension = (n+TILE_WIDTH_N-1)/TILE_WIDTH_N=2.

Each thread takes one M element and computes with each element of the tile (two rows of TILE_WIDTH_N elements: n1,…,n8) from N in shared memory.

Step 1

- Declare shared memory array for N elements with the size of TILE_WIDTH_RATIO_K×TILE_WIDTH_N.
  - __shared__ FLOAT_TYPE B_s[TILE_WIDTH_RATIO_K][TILE_WIDTH_N];

Step 2

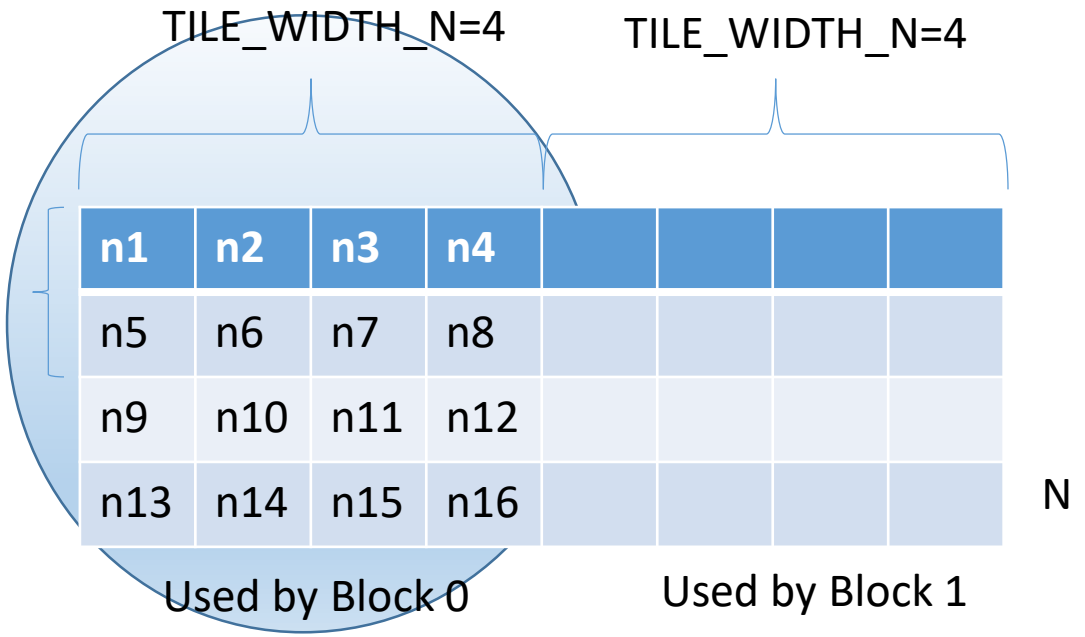
- Declare output array variable for P elements with the size of TILE_WIDTH_N and initialize the output array variable.
  - float P_reg [TILE_WIDTH_N];
  - //P_reg initialized to zero

Step 3

- Loop over the input tiles (the number of iterations = (k−1)/TILE_ WIDTH_RATIO_K+1, where k the number of the columns of matrix M
  - for(nIter = 0; n< (k−1)/TILE_ WIDTH_RATIO_K+1; nIter++) { … // see next slides}

Step 3.1

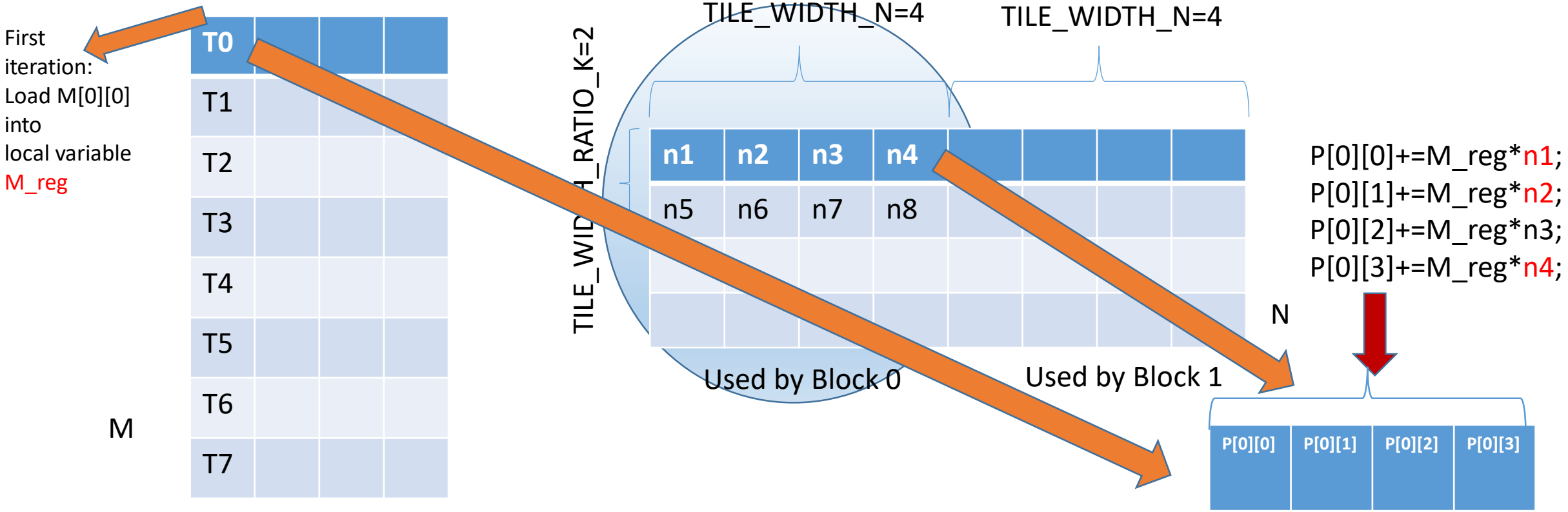
- Load the tile of N (size = TILE_WIDTH_RATIO_K × TILE_WIDTH_N) into shared memory. Note that one block has TILE_WIDTH_M threads, each loading one N element into shared memory.
- Please note that the block configuration is as follows:
  - dim3 dimBlock(TILE_WIDTH_M, 1);
- In this example, each iteration load two rows (since TILE_WIDTH_M=8, 8 N elements will be loaded into shared memory for each iteration). In block 0,
  - nIter=1 (1st iteration): load n1,…,n8
  - nIter=2 (2nd iteration): load n9,…,n16
- You need to figure out how to map the given threadIdx.x into the index of N elements (to be loaded into shared memory)



Step 3.2

M and P (before launching the kernel) should be row-major

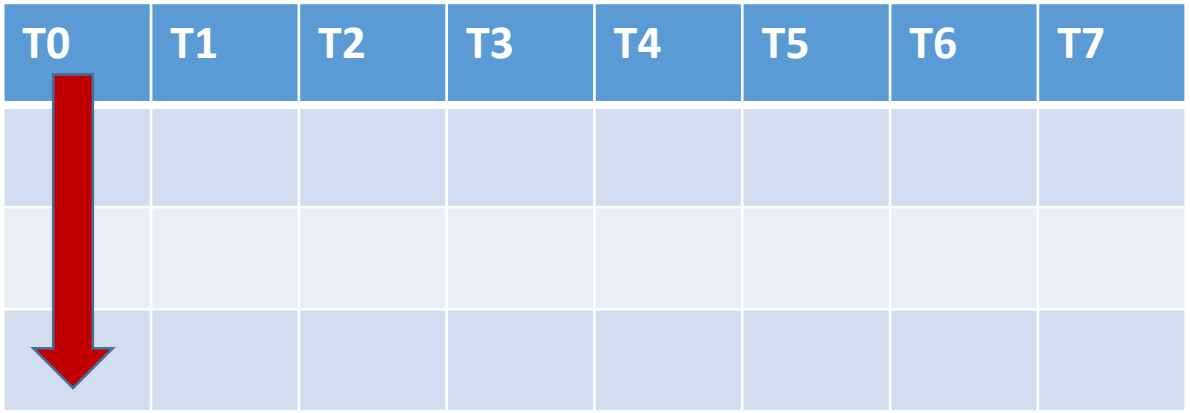
- Step 3.2 Loop over and update the output elements in the output array variable assigned to this thread. Note that output array variable is local variable.
  - for (uint cnt = 0; i < TILE_WIDTH_RATIO_K; ++cnt) { … }
- It accumulates the partial results. In this innerloop, the number of iteration is TILE_WIDTH_N=4. This is the first iteration (cnt = 0).



View of M’ and P’ (after launching the kernel)

- You should regard M and P in the previous slide as row-major
- While in the launched kernel, they should be column major (M’ and P’) as follows

M’

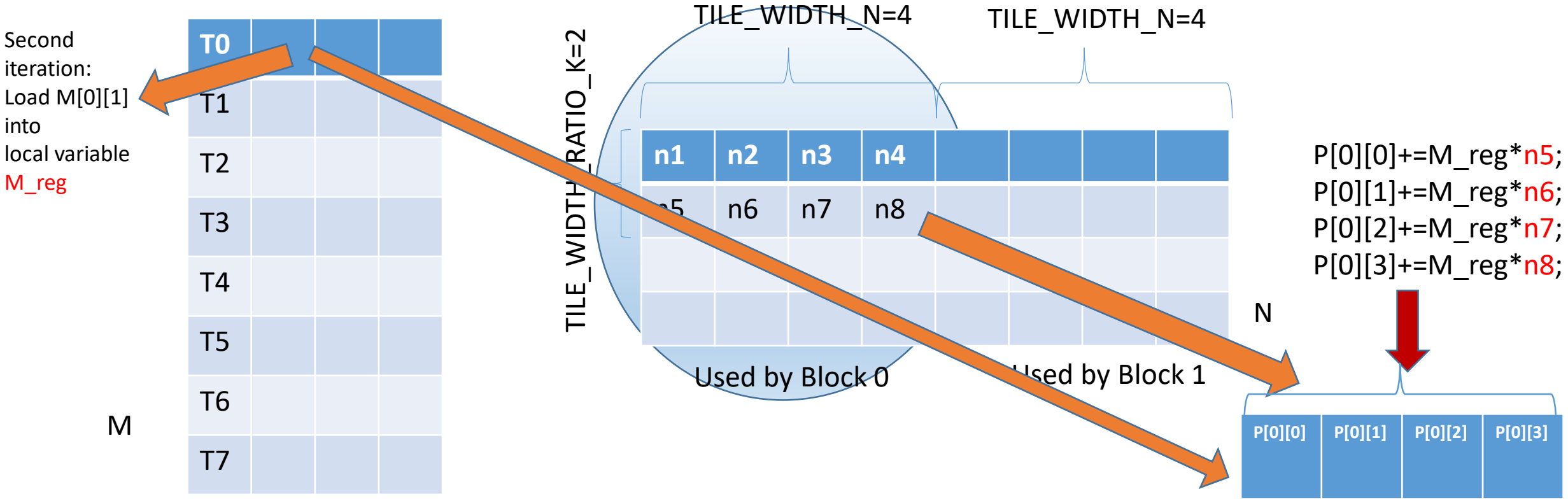


P’

P’[0][0]
P’[1][0]
P’[2][0]
P’[3][0]

Step 3.2 - Second iteration

- Step 3.2. This is the second iteration (cnt=1).



Step 4

- Store the output array variable to P elements (each thread stores TILE_WIDTH_N P elements and one block outputs TILE_WIDTH_N×TILE_WIDTH_M P elements
  - Grid configuraton:
    - dim3 dimGrid((m+TILE_WIDTH_M-1)/TILE_WIDTH_M, (n+TILE_WIDTH_N-1)/TILE_WIDTH_N);