GPU Teaching Kit

Accelerated Computing

Module 4.5 - Memory and Data Locality

Handling Arbitrary Matrix Sizes in Tiled Algorithms

Objective

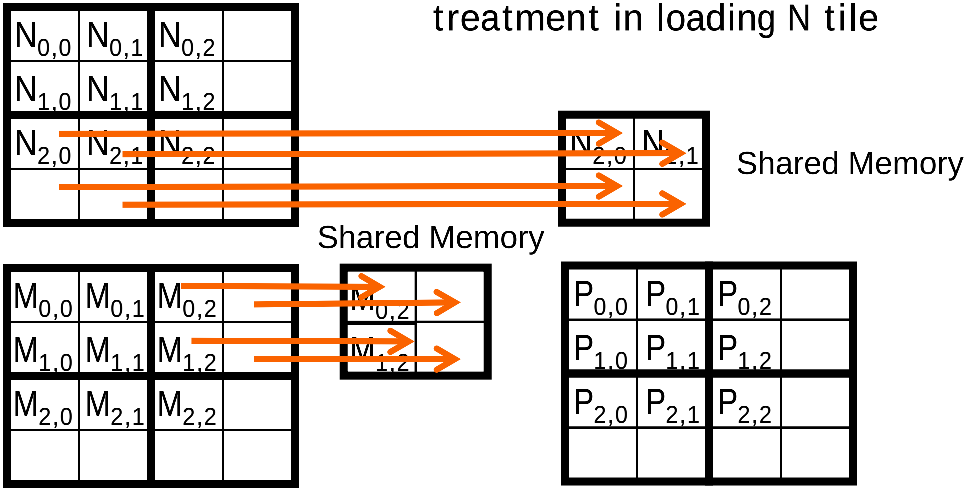
- To learn to handle arbitrary matrix sizes in tiled matrix multiplication
  - Boundary condition checking
  - Regularizing tile contents
  - Rectangular matrices

Handling Matrix of Arbitrary Size

- The tiled matrix multiplication kernel we presented so far can handle only square matrices whose dimensions (Width) are multiples of the tile width (TILE_WIDTH)
  - However, real applications need to handle arbitrary sized matrices.
  - One could pad (add elements to) the rows and columns into multiples of the tile size, but would have significant space and data transfer time overhead.
- We will take a different approach.

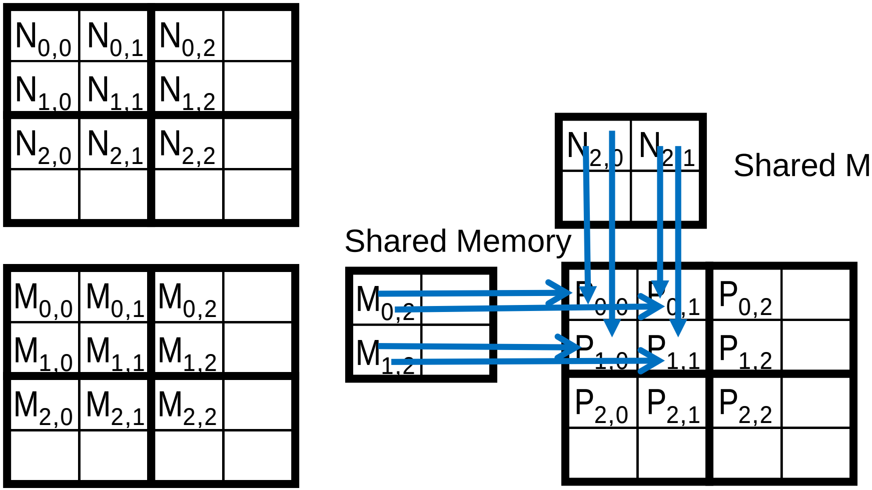
Phase 1 Loads for Block (0,0) for a 3x3 Example

Threads (1,0) and (1,1) need special treatment in loading N tile

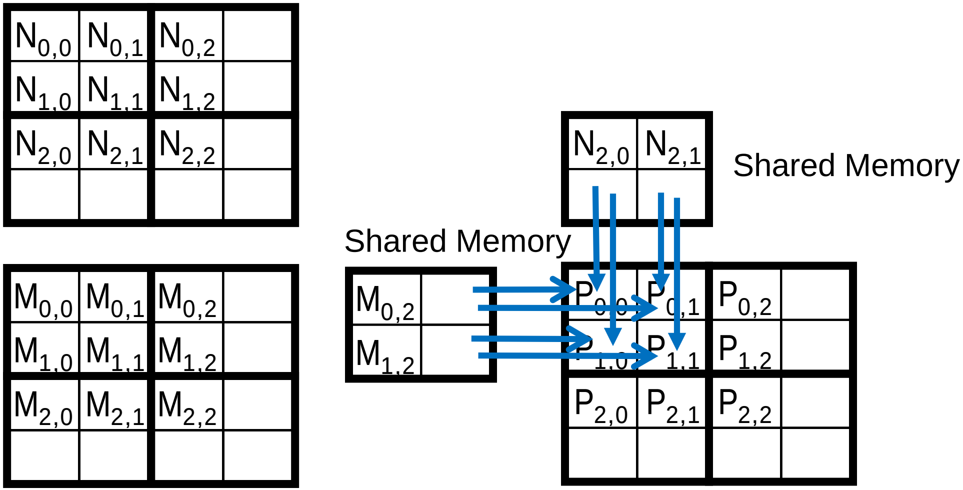


Threads (0,1) and (1,1) need special treatment in loading M tile

Phase 1 Use for Block (0,0) (iteration 0)



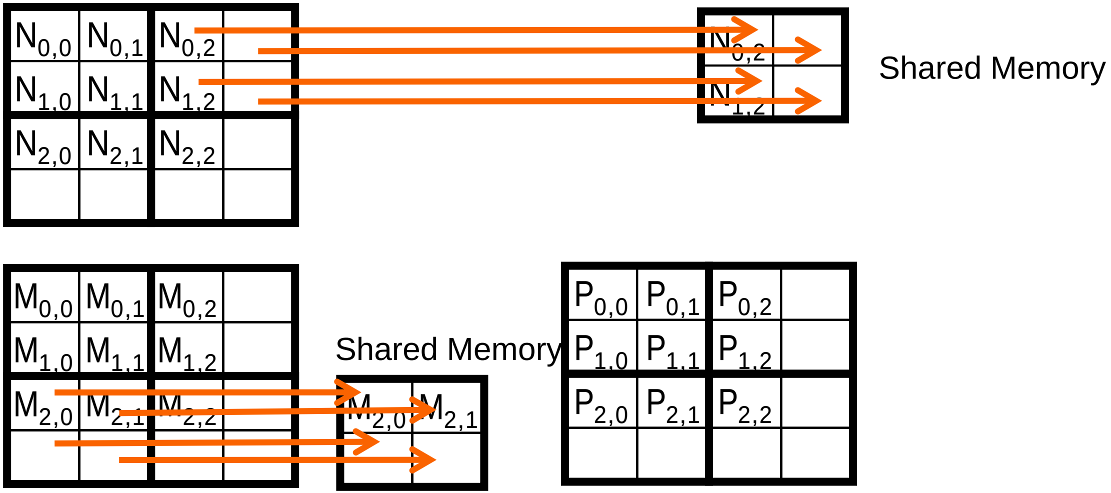
Phase 1 Use for Block (0,0) (iteration 1)



All Threads need special treatment. None of them should introduce invalidate contributions to their P elements.

Phase 0 Loads for Block (1,1) for a 3x3 Example

Threads (0,1) and (1,1) need special treatment in loading N tile



Threads (1,0) and (1,1) need special treatment in loading M tile

Major Cases in Toy Example

- Threads that do not calculate valid P elements but still need to participate in loading the input tiles
  - Phase 0 of Block(1,1), Thread(1,0), assigned to calculate non-existent P[3,2] but need to participate in loading tile element N[1,2]

- Threads that calculate valid P elements may attempt to load non-existing input elements when loading input tiles
  - Phase 0 of Block(0,0), Thread(1,0), assigned to calculate valid P[1,0] but attempts to load non-existing N[3,0]

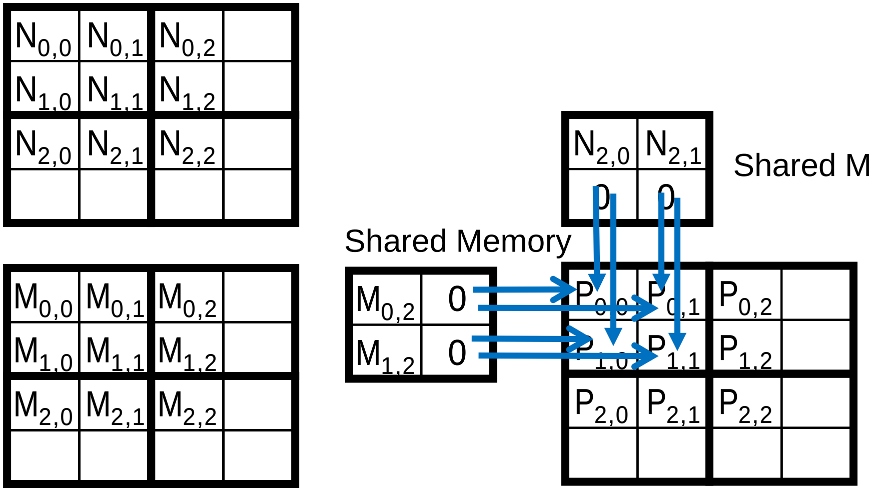
A "Simple" Solution

- When a thread is to load any input element, test if it is in the valid index range
  - If valid, proceed to load
  - Else, do not load, just write a 0

- Rationale: a 0 value will ensure that that the multiply-add step does not affect the final value of the output element

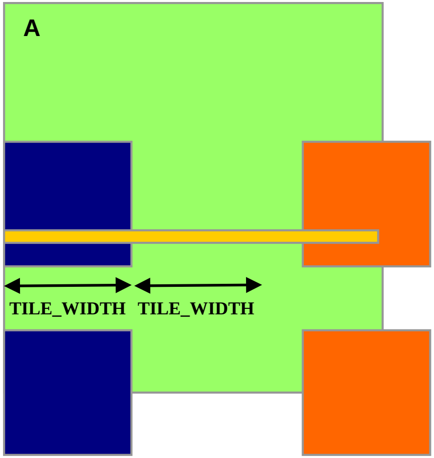
- The condition tested for loading input elements is different from the test for calculating output P element
  - A thread that does not calculate valid P element can still participate in loading input tile elements

Phase 1 Use for Block (0,0) (iteration 1)



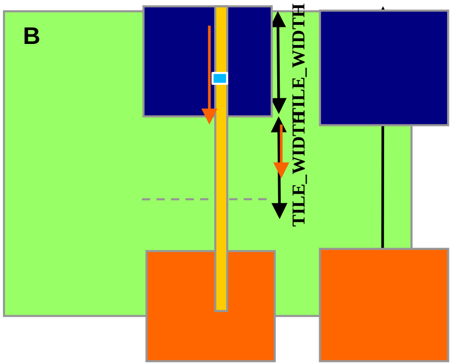
Boundary Condition for Input M Tile

- Each thread loads
  - M[Row][p*TILE_WIDTH+tx]
  - M[Row*Width + p*TILE_WIDTH+tx]
- Need to test
  - (Row < Width) && (p*TILE_WIDTH+tx < Width)
  - If true, load M element
  - Else , load 0



Boundary Condition for Input N Tile

- Each thread loads
  - N[p*TILE_WIDTH+ty][Col]
  - N[(p*TILE_WIDTH+ty)*Width+ Col]
- Need to test
  - (p*TILE_WIDTH+ty < Width) && (Col< Width)
  - If true, load N element
  - Else , load 0



Loading Elements – with boundary check

```
8    for (int p = 0; p < (Width-1) / TILE_WIDTH + 1; ++p) {

++       if(Row < Width && t * TILE_WIDTH+tx < Width) {
9            ds_M[ty][tx] = M[Row * Width + p * TILE_WIDTH + tx];
++       } else {
++           ds_M[ty][tx] = 0.0;
++       }
++       if (p*TILE_WIDTH+ty < Width && Col < Width) {
10           ds_N[ty][tx] = N[(p*TILE_WIDTH + ty) * Width + Col];
++       } else {
++           ds_N[ty][tx] = 0.0;
++       }
11       __syncthreads();
```

Inner Product – Before and After

- ++ if(Row < Width && Col < Width) {
- 12 for (int i = 0; i < TILE_WIDTH; ++i) {
- 13 Pvalue += ds_M[ty][i] * ds_N[i][tx];
- }
- 14 __syncthreads();
- 15 } /* end of outer for loop */
- ++ if (Row < Width && Col < Width)
- 16 P[Row*Width + Col] = Pvalue;
- } /* end of kernel */

Some Important Points

- For each thread the conditions are different for
  - Loading M element
  - Loading N element
  - Calculating and storing output elements
- The effect of control divergence should be small for large matrices

Handling General Rectangular Matrices

- In general, the matrix multiplication is defined in terms of rectangular matrices
  - A j x k M matrix multiplied with a k x l N matrix results in a j x l P matrix

- We have presented square matrix multiplication, a special case

- The kernel function needs to be generalized to handle general rectangular matrices
  - The Width argument is replaced by three arguments: j, k, l
  - When Width is used to refer to the height of M or height of P, replace it with j
  - When Width is used to refer to the width of M or height of N, replace it with k
  - When Width is used to refer to the width of N or width of P, replace it with l

GPU Teaching Kit

Accelerated Computing