How to Handle Matrices of Other Sizes?

- Lecture 5’s tiled kernel
  - assumed integral number of tiles (thread blocks)
  - in all matrix dimensions.
  
  How can we avoid this assumption?

- One answer: add padding, but not easy to reformat data, and adds transfer time.

Other ideas?

Let’s Review Our Kernel

```c
__global__ void MatrixMulKernel(float* M, float* N, float* P, int Width)
{
  __shared__ float subTileM[TILE_WIDTH][TILE_WIDTH];
  __shared__ float subTileN[TILE_WIDTH][TILE_WIDTH];
  int bx = blockIdx.x;  int by = blockIdx.y;
  int tx = threadIdx.x; int ty = threadIdx.y;
  // Identify the row and column of the P element to work on
  int Row = by * TILE_WIDTH + ty; // note: blockDim.x == TILE_WIDTH
  int Col = bx * TILE_WIDTH + tx; //       blockDim.y == TILE_WIDTH
  float Pvalue = 0;
  // Loop over the M and N tiles required to compute the P element
  // The code assumes that the Width is a multiple of TILE_WIDTH!
  for (int m = 0; m < Width/TILE_WIDTH; ++m) {
    subTileM[ty][tx] = M[Row*Width + m*TILE_WIDTH+tx];
    subTileN[ty][tx] = N[(m*TILE_WIDTH+ty)*Width+Col];
    __syncthreads();
    for (int k = 0; k < TILE_WIDTH; ++k)
      Pvalue += subTileM[ty][k] * subTileN[k][tx];
    __syncthreads();
  }
  P[Row*Width+Col] = Pvalue;
}
```
Second Tile Load for Block (0,0)

Second Tile Use for Block (0,0), k of 0

Second Tile Use for Block (0,0), k of 1

First Tile Load for Block (1,1)
Major Cases in Toy Example

- Threads that calculate valid P elements but can step outside valid input
  - Second tile of Block(0,0), all threads when k is 1
- Threads that do not calculate valid P elements
  - Block(1,1), Thread(1,0), non-existent row
  - Block(1,1), Thread(0,1), non-existing column
  - Block(1,1), Thread(1,1), non-existing row/column

Solution: Write 0 for Missing Elements

- Test during tile load: is target within input matrix?
  - If yes, proceed to load;
  - otherwise, just write 0 to shared memory.

- The benefit?
  - No specialization during tile use!
  - Multiplying by 0 guarantees that unwanted terms do not contribute to the inner product.
What About Threads Outside of P?

- If a thread is not within P,
  - All terms in sum are 0.
  - No harm in performing FLOPs.
  - No harm in writing to registers.

  - Must not be allowed to write to global memory!

So: Threads outside of P calculate 0, but store nothing.

Modifying the Tile Count

8. for (int m = 0; m < Width/TILE_WIDTH; ++m) {
    The bound for m implicitly assumes that Width is a multiple of TILE_WIDTH. We need to round up.
    for (int m = 0; m < (Width - 1)/TILE_WIDTH + 1; ++m) {
        For non-multiples of TILE_WIDTH:
        - quotient is unchanged;
        - add one to round up.
        For multiples of TILE_WIDTH:
        - quotient is now one smaller,
        - but we add 1.
Modifying the Tile Loading Code

We had …

// Collaborative loading of M and N tiles into shared memory
9. subTileM[ty][tx] = M[Row*Width + m*TILE_WIDTH+tx];
10. subTileN[ty][tx] = N[(m*TILE_WIDTH+ty)*Width+Col];

Note: the tests for M and N tiles are NOT the same.

if (Row < Width && m*TILE_WIDTH+tx < Width) {
    subTileM[ty][tx] = M[Row*Width + m*TILE_WIDTH+tx];
} else {
    subTileM[ty][tx] = 0;
}

© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007-2018 ECE408/CS483/
University of Illinois at Urbana-Champaign

Modifying the Tile Use Code

We had …

12. for (int k = 0; k < TILE_WIDTH; ++k)
13.     Pvalue += subTileM[ty][k] * subTileN[k][tx];

Note: no changes are needed, but we might save a little energy (fewer floating-point ops)?

if (Row < Width && Col < Width) {
    for (int k = 0; k < TILE_WIDTH; ++k)
        Pvalue += subTileM[ty][k] * subTileN[k][tx];
}

© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007-2018 ECE408/CS483/
University of Illinois at Urbana-Champaign

And for Loading N…

We had …

// Collaborative loading of M and N tiles into shared memory
9. subTileM[ty][tx] = M[Row*Width + m*TILE_WIDTH+tx];
10. subTileN[ty][tx] = N[(m*TILE_WIDTH+ty)*Width+Col];

Note: the tests for M and N tiles are NOT the same.

if (m*TILE_WIDTH+ty < Width && Col < Width ) {
    subTileN[ty][tx] = N[(m*TILE_WIDTH+ty)*Width+Col];
} else {
    subTileN[ty][tx] = 0;
}

© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007-2018 ECE408/CS483/
University of Illinois at Urbana-Champaign

Modifying the Write to P

We had …

16. P[Row*Width+Col] = Pvalue;

We must test for threads outside of P:

if (Row < Width && Col < Width) {
    P[Row*Width+Col] = Pvalue;
}

© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007-2018 ECE408/CS483/
University of Illinois at Urbana-Champaign
Some Important Points

• For each thread, conditions are different for
  – Loading M element
  – Loading N element
  – Calculation/storing output elements
• Branch divergence
  – affects only blocks on boundaries, and
  – should be small for large matrices.
• What about rectangular matrices?