Applied Parallel Programming

CNN

Objective

- To learn more about the implementation of a convolutional neural network
  - Levels of parallelism
  - Loop transformations
  - Basic kernel design

Sequential Code for the Forward Path of a Convolution Layer

```
void convLayer_forward(int M, int C, int H, int W, int K, float* X, float* W, float* Y)
{

  for(int m = 0;  m < M;  m++)                    // for each output feature map
     for(int h = 0; h < H_out; h++)                // for each output element
        for(int w = 0; w < W_out; w++) {
           Y[m, h, w] = 0;
           for(int c = 0;  c < C; c++)            // sum over all input feature maps
              for(int p = 0; p < K; p++)          // KxK  filter
                 for(int q = 0; q < K; q++)
                    Y[m, h, w] += X[c, h + p, w + q] * W[m, c, p, q];
        }
}
```

A Small Convolution Layer Example Generating Y[0,0,1]

X[0,_, _]

1	2	0	1
1	1	3	2
0	2	2	0
2	1	0	3

W[0,0,_, _]

1	1	1
2	2	3
2	1	0

X[1,_, _]

0	2	1	0
0	3	2	1
1	1	0	2
2	1	0	3

W[0,1,_, _]

1	2	3
1	1	0
3	0	1

Y[0,_, _]

0	?
?	?

X[2,_, _]

1	2	1	0
0	1	3	2
3	3	2	0
1	3	2	0

W[0,2,_, _]

0	1	1
1	0	2
1	2	1

A Small Convolution Layer Example
Generating Y[0,0,0], c=0

X[0,_, _]

1	2	0	1
1	1	3	2
0	2	2	0
2	1	0	3

W[0,0,_, _]

1	1	1
2	2	3
2	1	0

3+13+2

Y[0,_, _]

18	?
?	?

X[1,_, _]

0	2	1	0
0	3	2	1
1	1	0	2
2	1	0	3

W[0,1,_, _]

1	2	3
1	1	0
3	0	1

X[2,_, _]

1	2	1	0
0	1	3	2
3	3	2	0
1	3	2	0

W[0,2,_, _]

0	1	1
1	0	2
1	2	1

A Small Convolution Layer Example
Generating Y[0,0,0], c=1

X[0,_, _]

1	2	0	1
1	1	3	2
0	2	2	0
2	1	0	3

W[0,0,_, _]

1	1	1
2	2	3
2	1	0

7+3+3

X[1,_, _]

0	2	1	0
0	3	2	1
1	1	0	2
2	1	0	3

W[0,1,_, _]

1	2	3
1	1	0
3	0	1

Y[0,_, _]

31	?
?	?

X[2,_, _]

1	2	1	0
0	1	3	2
3	3	2	0
1	3	2	0

W[0,2,_, _]

0	1	1
1	0	2
1	2	1

A Small Convolution Layer Example Generating Y[0,0,0], c=2

X[0,_, _]

1	2	0	1
1	1	3	2
0	2	2	0
2	1	0	3

W[0,0,_, _]

1	1	1
2	2	3
2	1	0

X[1,_, _]

0	2	1	0
0	3	2	1
1	1	0	2
2	1	0	3

W[0,1,_, _]

1	2	3
1	1	0
3	0	1

Y[0,_, _]

51	?
?	?

3+6+11

X[2,_, _]

1	2	1	0
0	1	3	2
3	3	2	0
1	3	2	0

W[0,2,_, _]

0	1	1
1	0	2
1	2	1

Parallelism in a Convolution Layer

- All output feature maps can be calculated in parallel
  - A small number in general, not sufficient to fully utilize a GPU
- All output feature map pixels can be calculated in parallel
  - All rows can be done in parallel
  - All pixels in each row can be done in parallel
  - Large number but diminishes as we go into deeper layers
- All input feature maps can be processed in parallel, but will need atomic operation or tree reduction

Design of a Basic Kernel

- Each block computes a tile of output pixels
  - TILE_WIDTH pixels in each dimension
- The first (x) dimension in the grid maps to the M output feature maps
- The second (y) dimension in the grid maps to the tiles in the output feature maps

Host Code for the Basic Kernel

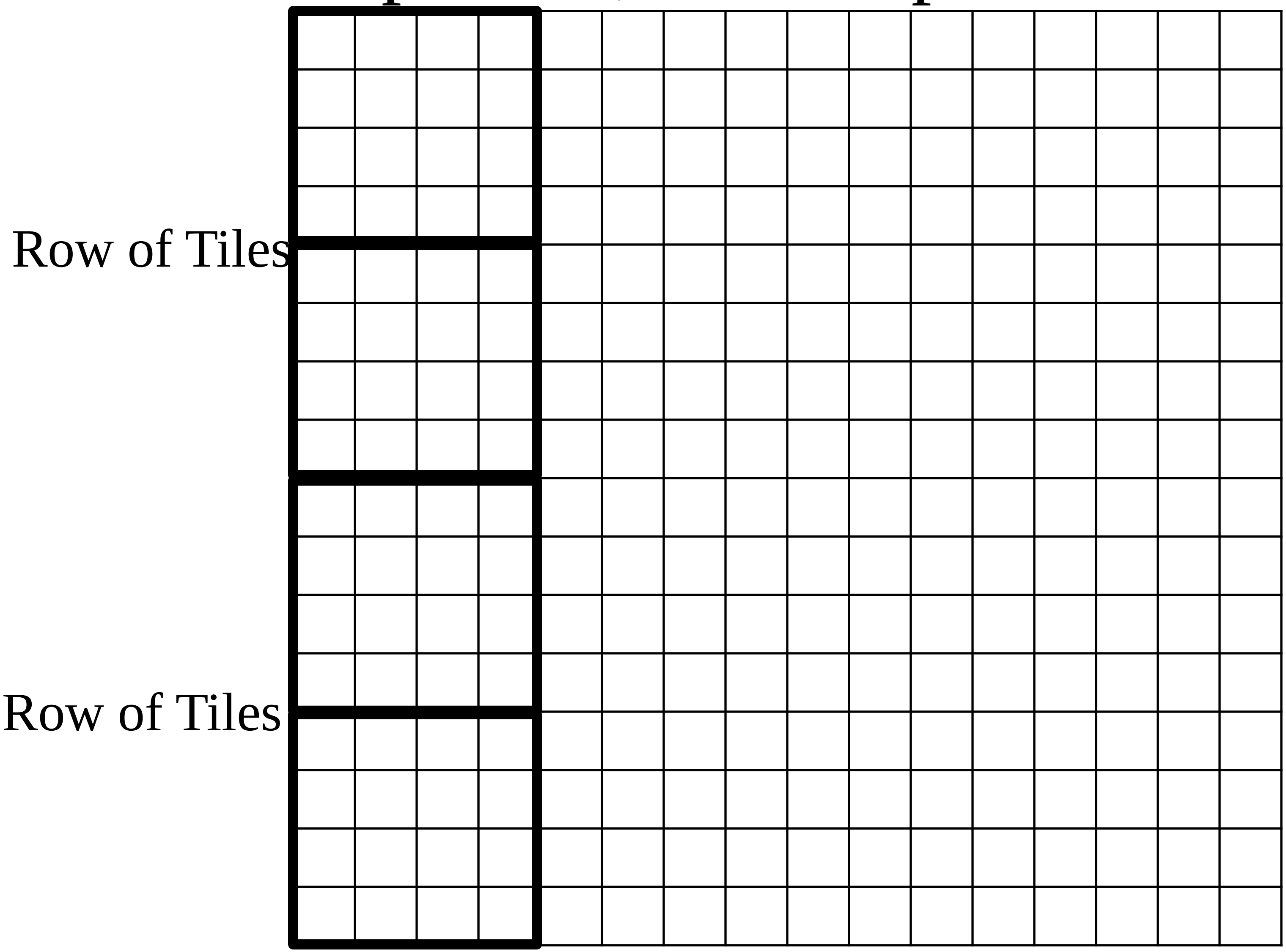
- Defining the grid configuration
  - W_out and H_out are the output feature map width and height

```
# define TILE_WIDTH 16          // We will use 4 for small examples.
W_grid = W_out/TILE_WIDTH; // number of horizontal tiles per output map
H_grid = H_out/TILE_WIDTH;  // number of vertical tiles per output map
Y = H_grid * W_grid;
dim3 blockDim(TILE_WIDTH, TILE_WIDTH, 1);
dim3 gridDim(M, Y, 1);
ConvLayerForward_Kernel<<< gridDim, blockDim>>>(…);
```

A Small Example

- Assume that we will produce 4 output feature maps
  - Each output feature map is 8x8 image
  - We have 4 blocks in the x dimension
- If we use tiles of 4 pixels on each side (TILE_SIZE = 4)
  - We have 4 blocks in the x dimension
    - Top two blocks in each column calculates the top row of tiles in the corresponding output feature map
    - Bottom two block in each column calculates the bottom row of tiles in the corresponding output feature map

Mapping Threads to Output Feature Maps Grid Perspective, first output feature mapp



A Basic Conv. Layer Forward Kernel (Code is incomplete!)

```
__global__ void ConvLayerForward_Basic_Kernel(int C, int W_grid, int K,
            float* X, float* W, float* Y)
{
   int m = blockIdx.x;
   int h =  blockIdx.y / W_grid  + threadIdx.y;
   int w = blockIdx.y % W_grid + threadIdx.x;
   float acc = 0.;
   for (int c = 0;  c < C; c++) {                        // sum over all input channels
     for (int p = 0; p < K; p++)                         // loop over KxK  filter
       for (int q = 0; q < K; q++)
         acc += X[c, h + p, w + q] * W[m, c, p, q];
   }
   Y[m, h, w] = acc;
}
```

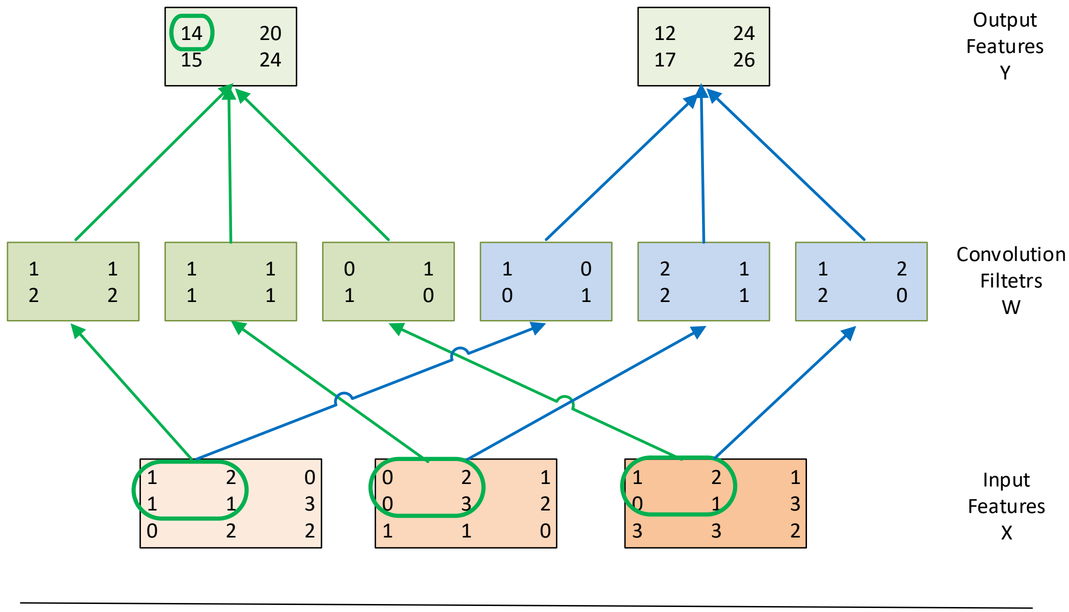
Some Observations

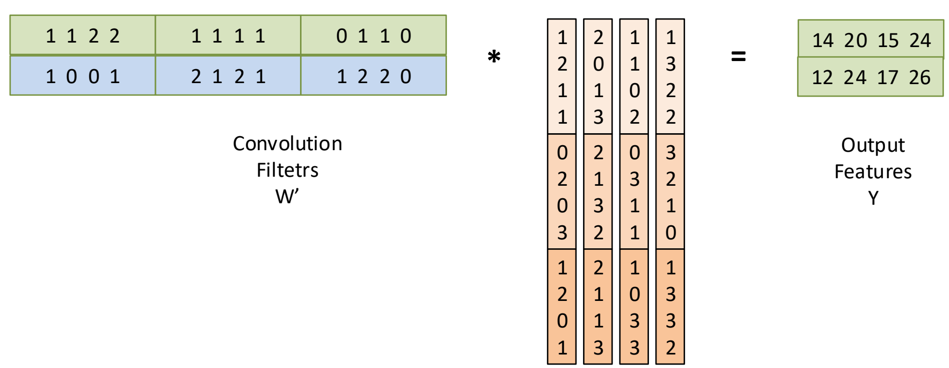
- The amount of parallelism is quite high as long as the total number of pixels across all output feature maps is large
  – This matches the CNN architecture well
- Each input tile is loaded multiple times, once for each block that calculates the output tile that requires the input tile
  – Not very efficient in global memory bandwith

Sequential code for the Forward Path of a Sub-sampling Layer

```
void poolingLayer_forward(int M, int H, int W, int K, float* Y, float* S)
{
  for(int m = 0;  m < M;  m++)                    // for each output feature maps
    for(int h = 0; h < H/K; h++)                  // for each output element
      for(int w = 0; w < W/K; w++) {
        S[m, x, y] = 0.;
        for(int p = 0; p < K; p++) {               // loop over KxK input samples
          for(int q = 0; q < K; q++)
            S[m, h, w] += Y[m, K*h + p, K*w + q] /(K*K);
        }
        // add bias and apply non-linear activation
        S[m, h, w] = sigmoid(S[m, h, w] + b[m])
      }
}
```

Implementing a convolution layer with matrix multiplication





Input Features X_unrolled

ANY QUESTIONS?