ECE408/CS483/CSE408 Spring 2020
Optimizing Convolution Layers

Contributed by Carl Pearson
pearson@illinois.edu

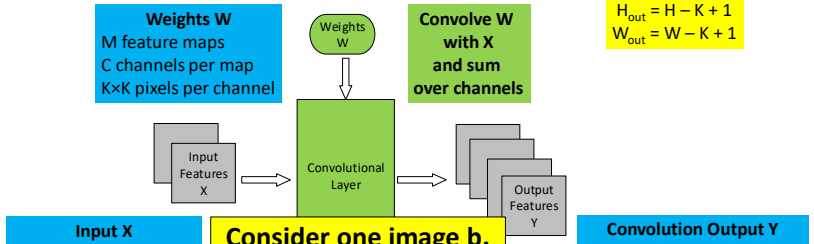
1

Objective

- To understand how unrolling input X can improve performance for convolution layers on GPUs.

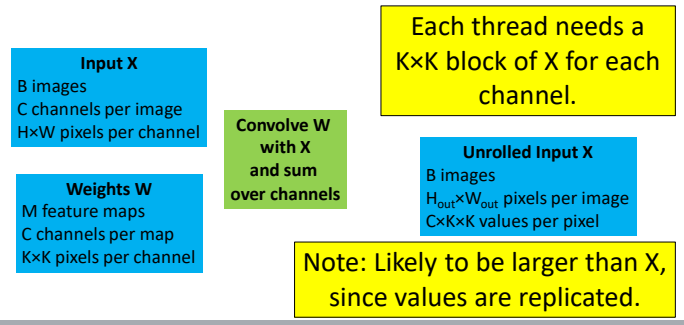
2

Reorganize Input X for Convolution



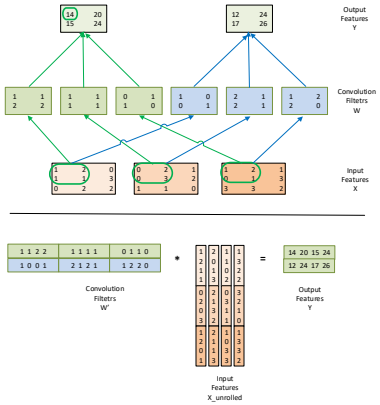
3

Reorganize Input X for Convolution



4

Implementing a Convolution Layer with Matrix Multiplication

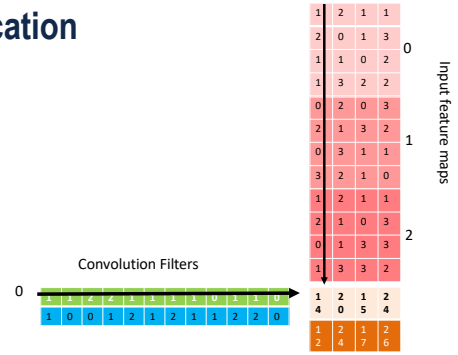


5

Simple Matrix Multiplication

Each product matrix element is an output feature map pixel.

This inner product generates element 0 of output feature map 0.

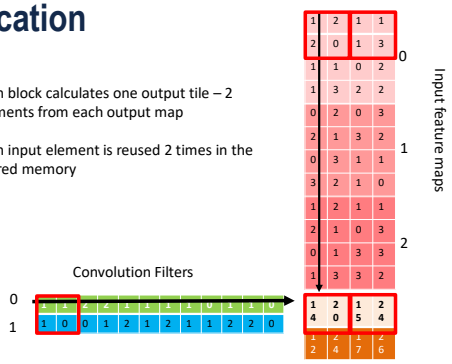


6

Tiled Matrix Multiplication 2x2 Example

Each block calculates one output tile – 2 elements from each output map

Each input element is reused 2 times in the shared memory

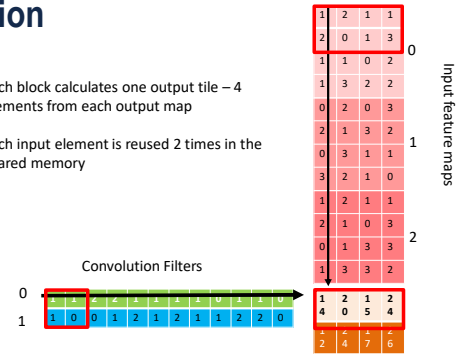


7

Tiled Matrix Multiplication 2x4 Example

Each block calculates one output tile – 4 elements from each output map

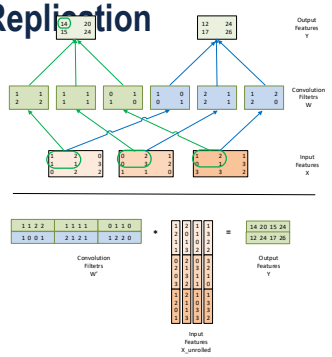
Each input element is reused 2 times in the shared memory



8

Efficiency Analysis: Total Input Replication

- Replicated input features are shared among output maps
  - There are H_out * W_out output feature map elements
  - Each requires K*K elements from the input feature maps
  - So, the total number of input element after replication is H_out*W_out*K*K times for each input feature map
  - The total number of elements in each original input feature map is (H_out+K-1) * (W*out+K-1)



9

Analysis of a Small Example

H_out = 2

W_out = 2

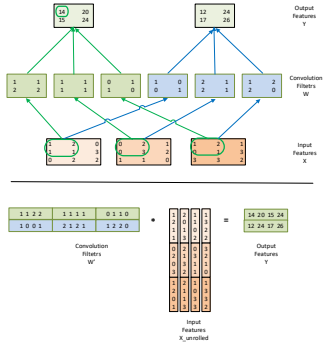
K = 2

There are 3 input maps (channels)

The total number of input elements in the replicated (“unrolled”) input matrix is 3*2*2*2*2

The replicating factor is

(3*2*2*2*2)/(3*3*3) = 1.78



10

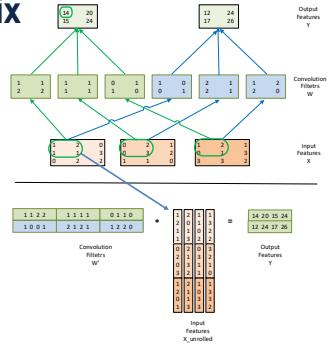
Memory Access Efficiency of Original Convolution Algorithm

- Assume that we use tiled 2D convolution
- For input elements
  - Each output tile has TILE_WIDTH^2 elements
  - Each input tile has (TILE_WIDTH+K-1)^2
  - The total number of input feature map element accesses was TILE_WIDTH^2*K^2
  - The reduction factor of the tiled algorithm is K^2*TILE_WIDTH^2/(TILE_WIDTH+K-1)^2
- The convolution filter weight elements are reused within each output tile

11

Properties of the Unrolled Matrix

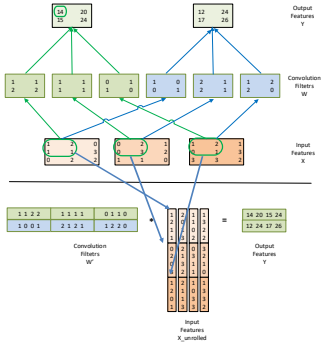
- Each unrolled column correspond to an output feature map element
- For an output feature element (h,w), the index for the unrolled column is h*W_out+w (linearized index of the output feature map element)



12

Properties of the Unrolled Matrix (cont.)

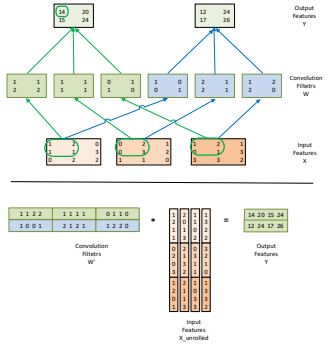
- Each section of the unrolled column correspond to an input feature map
- Each section of the unrolled column has k*k elements (convolution mask size)
- For an input feature map c, the vertical index of its section in the unrolled column is c*k*k(linearized index of the output feature map element)



13

To Find the Input Elements

- For output element (h,w), the base index for the upper left corner of the input feature map c is (c, h, w)
- The input element index for multiplication with the convolution mask element (p, q) is (c, h+p, w+q)



14

Input to Unrolled Matrix Mapping

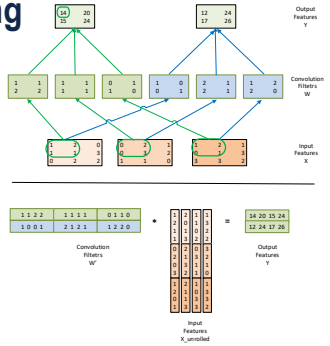
```
Output element (h, w)
Mask element (p, q)
Input feature map c

// calculate the horizontal matrix index
int w_unroll = h * W_out + w;

// find the beginning of the unrolled
int w_base = c * (K*K);

// calculate the vertical matrix index
int h_unroll = w_base + p * K + q;

X_unroll[b, h_unroll, w_unroll] = X[b, c, h + p, w + q];
```



15

Function to generate "unrolled" X

```
void unroll(int B, int C, int H, int W, int K, float* X, float* X_unroll)  {
  int H_out = H - K + 1;                        // calculate H_out, W_out
  int W_out = W - K + 1;
  for (int b = 0; b < B; ++b)                   // for each image
    for (int c = 0; c < C; ++c) {               // for each input channel
      int w_base = c * (K*K);                   // per-channel offset for smallest X_unroll index
      for (int p = 0; p < K; ++p)               // for each element of KxK filter (two loops)
        for (int q = 0; q < K; ++q) {
          for (int h = 0; h <  H_out; ++h)      // for each thread (each output value, two loops)
            for (int w = 0; w < W_out; ++w) {
              int h_unroll = w_base + p * K + q;  // data needed by one thread
              int w_unroll = h * W_out + w;       // smallest index--across threads (output values)
              X_unroll[b, h_unroll, w_unroll] = X[b, c, h + p, w + q]; // copy input pixels
            }
    }
}
```


16

Implementation Strategies for a Convolution Layer

- Baseline
  - Tiled 2D convolution implementation, use constant memory for convolution masks
- Matrix-Multiplication Baseline
  - Input feature map unrolling kernel, constant memory for convolution masks as an optimization
  - Tiled matrix multiplication kernel
- Matrix-Multiplication with built-in unrolling
  - Perform unrolling only when loading a tile for matrix multiplication
  - The unrolled matrix is only conceptual
  - When loading a tile element of the conceptual unrolled matrix into the shared memory, use the properties in the lecture to load from the input feature map
- More advanced Matrix-Multiplication
  - Use joint register-shared memory tiling