L9: Convolution

Cong (Callie) Hao
callie.hao@ece.gatech.edu

Assistant Professor
ECE, Georgia Institute of Technology

Sharc-lab @ Georgia Tech https://sharclab.ece.gatech.edu/

Acknowledgement: some contents are borrowed from Prof. Bei Yu at The Chinese University of Hong Kong
• Basic Convolution Computation
• Data Reuse: Input/Output/Weight stationary
• Img2Col and advanced Img2Col
• Sparse Matrix Multiplication
CNN Computation with Multiple Channels

```
for(int h = 0; h < H-2; h++) {
    for(int w = 0; w < W-2; w++) {
        float sum = 0;
        for(int ci = 0; ci < CI; ci++) {
            for(int m = 0; m < K; m++) {
                for(int n = 0; n < K; n++) {
                    sum += A[ci][h+m][w+n] * W[ci][m][n];
                }
            }
            B[h][w] = sum;
        }
    }
}
```
CNN Computation with Multiple Channels

\[
\text{for(int } h = 0; h < H-2; h++) \{
  \text{for(int } w = 0; w < W-2; w++) \{
    \text{float sum = 0;}
    \text{for(int } ci = 0; ci < CI; ci++) \{
      \text{for(int } m = 0; m < K; m++) \{
        \text{for(int } n = 0; n < K; n++) \{
          \text{sum += } A[ci][h+m][w+n] \times W[ci][m][n];
        \}\}
      \}\}
    \text{B[h][w] = sum;}
  \}\}
\]
CNN Computation with Multiple Kernels

Input

Kernels (2)

Output channel 1

Output channel 2

CO=2
CNN Computation with Multiple Kernels

How do we explore parallelism?
Convolution Acceleration

• Parallelism 1: across output channel (kernel)

Naive: $H \times W \times CI \times K \times K \times CO$

Speedup: CO

In parallel

how to implement it in HLS?
Convolution Acceleration

- Parallelism 2: across input channel

Naive:
\[ H \times W \times CI \times K \times K \times CO \]

Speedup:
\[ CO \times CI \]

Partial sum
Convolution Acceleration

- Parallelism 3: across width dimension

Naive: $H \times W \times CI \times K \times K \times CO$

Speedup: $CO \times CI \times W$

But... pay attention to scalability
Both memory and computation can explode under aggressive parallelism

**Approaches**

- Try not partition more than 2 dimensions
- Consider tiling: cut the input image and intermediate feature maps into small cubes
  - Need to **consider tile boundary** – significantly affect the accuracy
- Parallelize at PE-granularity

**Read more:**

- Parallelize at PE level:
  - [https://sharc-knowledgebase.netlify.app/articles/pynq/u50/multiple_kernel_execution_demo/](https://sharc-knowledgebase.netlify.app/articles/pynq/u50/multiple_kernel_execution_demo/)
- Convolution tiling:
  - [https://sharc-knowledgebase.netlify.app/articles/cnn/tiling-based_convolution_for_hls/](https://sharc-knowledgebase.netlify.app/articles/cnn/tiling-based_convolution_for_hls/)
- Thanks to Akshay Kamath @ Sharc
Cutting One Layer into Tiles

• 8 Input Tiles, 12 Output Tiles

Output Tile 1  Tile 2  Tile 3
(2 channels)
Layer Padding and Tile Padding

- Layer Padding: zero padding

6x6 input  3x3 kernel
Layer Padding and Tile Padding

• Layer Padding: zero padding

6x6 input  3x3 kernel  4x4 output
Layer Padding and Tile Padding

- **Layer Padding:** zero padding

![Diagram showing zero padding with a 6x6 input, a 3x3 kernel, and a 4x4 output. The output is 6x6 after padding.]
Layer Padding and Tile Padding

- Tile Padding: dealing with tile boundary

One tile
Layer Padding and Tile Padding

- **Tile Padding**: dealing with tile boundary

- **Zero padding**

- **Tile boundary must be taken care of**
Data Reuse: Optimize Latency & Energy

• Data movement is **slow and expensive** (energy)
  - Once we load the data from DRAM to BRAM, let’s try to use them as much as possible

• Ways of reusing data in BRAM:
  - Input stationary
  - Weight stationary
  - Output stationary
  - Note: the “stationary” definition here is a little bit different as in systolic array
Input Stationary

• Assume only 2 tiles (1-in, 1-out) and a few kernels can fit into BRAM
Input Stationary

- Assume only 2 tiles (1-in, 1-out) and a few kernels can fit into BRAM

Load/store into/from BRAM and used for computation

Next: which tile(s) to swap out?
Input Stationary

• Try to keep the input tile(s) in BRAM as long as possible

Load/store into/from BRAM and used for computation
Try to keep the **input tile(s)** in BRAM as long as possible

- **Load/store into/from BRAM and used for computation**
Output Stationary

- Try to keep the output tile(s) in BRAM as long as possible

Loaded into BRAM and used for computation
Output Stationary

• Try to keep the output tile(s) in BRAM as long as possible

Loaded into BRAM and used for computation
Which is Better?

• Good question... I personally like output stationary
• Design space exploration (DSE)
  o Depending on your tile size, partition scheme, etc.
  o Please try to calculate if you’re interested
• Read more:
Outline

• Basic Convolution Computation
• Data Reuse: Input/Output/Weight stationary
• Img2Col and advanced Img2Col
• Sparse Matrix Multiplication
### From Convolution to Matrix Multiplication

- **Img2Col** (image-to-column) convolution – GEMM (General Matrix Multiply)

![Diagram](image)

<table>
<thead>
<tr>
<th>Callie Hao</th>
<th>Sharc-lab @ Georgia Institute of Technology</th>
<th><a href="https://sharclab.ece.gatech.edu/">Website</a></th>
<th>26</th>
</tr>
</thead>
</table>
From Convolution to Matrix Multiplication

\[ X \in \mathbb{R}^{d \times (k^2c)} \times W \in \mathbb{R}^{(k^2c) \times n} = Y \in \mathbb{R}^{d \times n} \]

Filters: \( n \times c \times k \times k \)

Reorder
Img2Col Pros and Cons

• Pros
  o **Good performance** and easy to implement (especially on GPUs)
  o Applicable for any convolution configuration on any platform
  o BLAS-friendly memory layout to enjoy SIMD/locality/parallelism

• Cons
  o **Large extra memory** overhead
Memory-efficient Convolution

- An improvement: remove “spatial” redundancy
  - Smaller memory footprint, cache locality, and explicit parallelism

DNNs may be redundant (over-parameterized), and filters may be sparse.
Sparse Matrix Representation

A matrix example

<table>
<thead>
<tr>
<th>A</th>
<th>rowptr</th>
<th>colptr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 3 0</td>
<td>row0 (3,2)</td>
<td>col0 (7,1), (6,3), (2,4)</td>
</tr>
<tr>
<td>7 0 0 0</td>
<td>row1 (7,0)</td>
<td>col1 (5,3)</td>
</tr>
<tr>
<td>0 0 4 8</td>
<td>row2 (4,2), (8,3)</td>
<td>col2 (3,0), (4,2), (3,3)</td>
</tr>
<tr>
<td>6 5 3 0</td>
<td>row3 (6,0), (5,1), (3,2)</td>
<td>col3 (8,2), (1,4), (8,5)</td>
</tr>
<tr>
<td>2 0 0 1</td>
<td>row4 (2,0), (1,3)</td>
<td></td>
</tr>
<tr>
<td>0 0 0 8</td>
<td>row5 (8,3)</td>
<td></td>
</tr>
</tbody>
</table>

Compressed Sparse Row (CSR)

Compressed Sparse Column (CSC)
Related Papers


- ...
Summary

• Basic Convolution Computation
  o Parallelism across Height, Width, Channel, Kernel, ...
  o Scalability is an issue, and tiling is almost unavoidable

• Data Reuse: Input/Output/Weight stationary
  o Depends on how you partition your tiles

• Img2Col and advanced Img2Col
  o Good for regular memory and GPU matrix multiplication but bad for memory

• Sparse Matrix Multiplication

• Converting Convolution to Matrix Multiplication can explore sparsity but has to pay extra cost