Symbolic Compiler: State-of-Art CNN implementation on FPGA

Sixiao Zhu, Ningyi Xu, MSRA

4/6/2016

Content

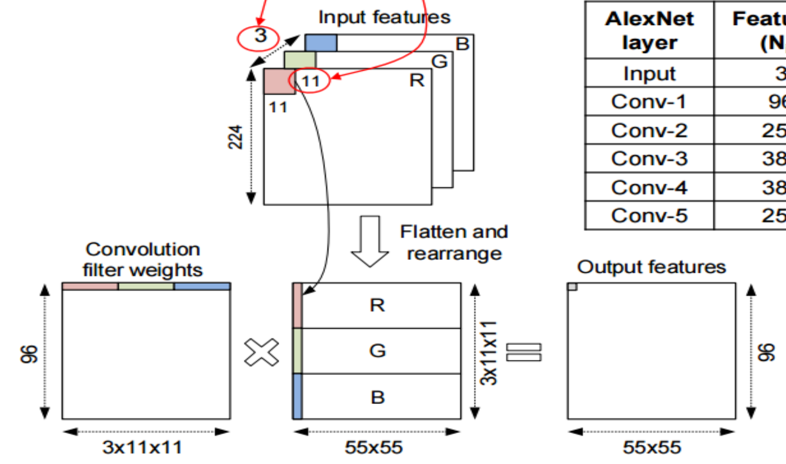
- A analysis of current difficulties to implement large CNN models on FPGA card.
- A novel solution to maximize IO bandwidth and put the performance bottleneck to calculation resources.
- Verified experiment result at VGG19 model that achieves 399.6 gops, 2.9 times faster than current known best related work.

Convolution as GEMM: Common idea

- Used by main stream CNN implementations: cuCNN, caffe.

3-D Convolution as Matrix Mult.

$$out(f_o,x,y)=\sum_{f_i=0}^{N_{if}}\sum_{k_x=0}^{K}\sum_{k_y=0}^{K} wt(f_o,f_i,k_x,k_y)\times in(f_i,x+k_x,y+k_y)$$



AlexNet layer	Features (N_{if})	Size	Kernel (K)	Stride
Input	3	224x224		
Conv-1	96	55x55	11x11	4
Conv-2	256	27x27	5x5	1
Conv-3	384	13x13	3x3	1
Conv-4	384	13x13	3x3	1
Conv-5	256	13x13	3x3	1

K. Chellapilla, et al. High performance convolutional neural networks for document processing. IWFHR 2006

Why not FFT?

- **Small filter size**. With FFT, one convolution at time domain equals one product at frequency domain. Small filter kernel gains less acceleration than big kernel, unfortunately, 3 is the most common case (VGG, AlexNet).
- **Complex multiplication consume 4 times more.** But the DSP resource are limited (now only 9/4 acceleration).
- **DFT and reverse DFT overhead**. Non-trivial, hard to parallel.
- **IO bottom neck.** Point wise multiplication at frequency domain relies heavily on DDR bandwidth (fetching : calculation = 1 : 1), 8GB/S DDR on FPGA (512bit / cycle) cannot keep up with calculation (up to 1024 multiplication / cycle).
- **In conclusion: gflops is even worse than GEMM based approach for 3 x 3 kernel convolution.**

Convolution as GEMM: Difficulties

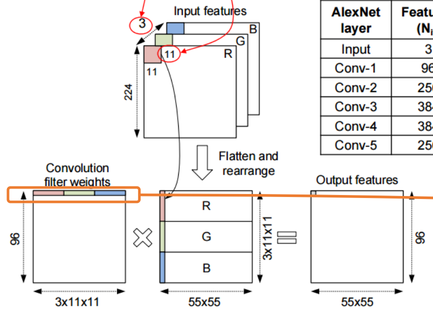
- **Model too big for block RAM.** 5.25MB BRAM for Stratix-V. In VGG19, maximal feature map size up to 3.2 MB, note input feature map and output feature map must simultaneously be stored on BRAM.
- **DDR access must be sequential.** 1024 DSP GEMM requires about 16 fetch per cycle. Through maximal DDR bandwidth achieves 8GB/S, that is about 512bit / cycle, but this bandwidth requires strict condition: Access must be 512bits at a time for a burst transfer, and access must hit sequential DDR address, otherwise DDR must recharge for random or large stride access, making IO a bottleneck.

Convolution as GEMM: Difficulties

- Online address convert not feasible.
    - Filter slide window covers 2D rectangle, can not be sequentially fetched . If kernel size == 3, every 3 numbers’ fetch causes a DDR recharge.
    - Successive element on target matrix randomly distributed across source feature maps, need to calculate actual address on feature map online, in parallel. Large resource consumption, and impossible to do parallel random access to DDR

3-D Convolution as Matrix Mult.

$$out(f_o,x,y)=\sum_{f_i=0}^{N_{if}}\sum_{k_x=0}^{K}\sum_{k_y=0}^{K} wt(f_o,f_i,k_x,k_y)\times in(f_i,x+k_x,y+k_y)$$



AlexNet layer	Features (N_{if})	Size	Kernel (K)	Stride
Input	3	224x224		
Conv-1	96	55x55	11x11	4
Conv-2	256	27x27	5x5	1
Conv-3	384	13x13	3x3	1
Conv-4	384	13x13	3x3	1
Conv-5	256	13x13	3x3	1

K. Chellapilla, et al. High performance convolutional neural networks for document processing. IWFHR 2006

Not the same #feature map:
3 V.S. 64

```
-1 -1 -1 -1 -1 -1 -1 -1
-1  0  1  2  3  4  5 -1
-1  6  7  8  9 10 11 -1
-1 12 13 14 15 16 17 -1
-1 18 19 20 21 22 23 -1
-1 24 25 26 27 28 29 -1
-1 30 31 32 33 34 35 -1
-1 -1 -1 -1 -1 -1 -1 -1
```

The first fmap, totally 64 of them.

64 X 3 X 3

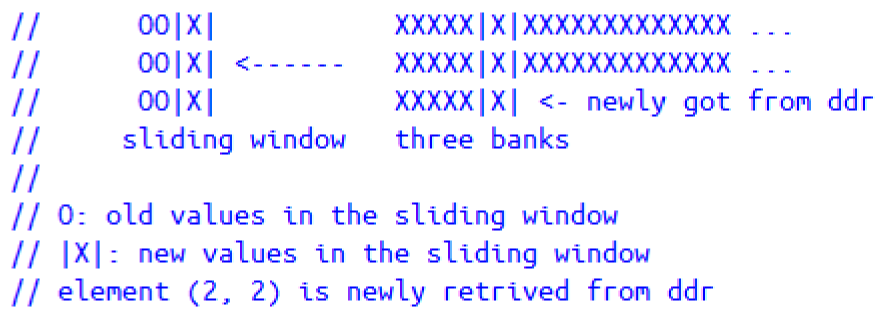
```
   0    1    2    6    7    8   12   13   14   36 ...  222  223  224  228  229  230  252
 253  254  258  259  260  264  265  266  288  289 ...  475  476  480  481  482  504  505
 506  510  511  512  516  517  518  540  541  542 ...  728  732  733  734  756  757  758
.
.
.
2030 2052 2053 2054 2058 2059 2060 2064 ... 2274 2275 2276 2280 2281 2282
```

Address on feature map storage in DDR

64 feature map's certain filter window location expand to one row of target matrix of length 64 X 3 X 3, successive 64 8bit fix number (burst length) strides across 7 ~ 8 feature maps and three rows of each feature map, thus the actual address of successive elements are nearly randomly distributed.

Convolution as GEMM: Difficulties

- Line buffer styled im2col is also not feasible.
  - $K \times K$ element per cycle, 512bit/cycle DDR not fully used for $K == 3$. im2col itself consumes more time than GEMM. Slow down system by > 50%.
  - Can't overlap execution with GEMM. DDR access conflicts.
  - Still, cannot generate the layout that GEMM requires. GEMM fetches matrix in a tiling manner, meanwhile line buffer generates each feature map



```
//      OO|X|          XXXXX|X|XXXXXXXXXXXXX ...
//      OO|X| <------  XXXXX|X|XXXXXXXXXXXXX ...
//      OO|X|          XXXXX|X| <- newly got from ddr
//     sliding window   three banks
//
// O: old values in the sliding window
// |X|: new values in the sliding window
// element (2, 2) is newly retrived from ddr
```

Solution: Change the feature map and filter memory format

```
L = size of one burst DDR read (512bit), also size of one
tile (64 8bit fix point number).

Old: fmap[Nfmap][Nposition]
// format protocal for both GEMM’s input and output (next
layer’s input)
New: fmap[Nfmap/L][Nposition][L]

Old: filter[Nfilter][Nposition]
// rearranged off line
New: filter[Nfilter/L][Nposition][L]
```

Basic Idea:
A memory layout protocol for both input fmap and output fmap.

$L_{fmap} = 6$

$K = 3$

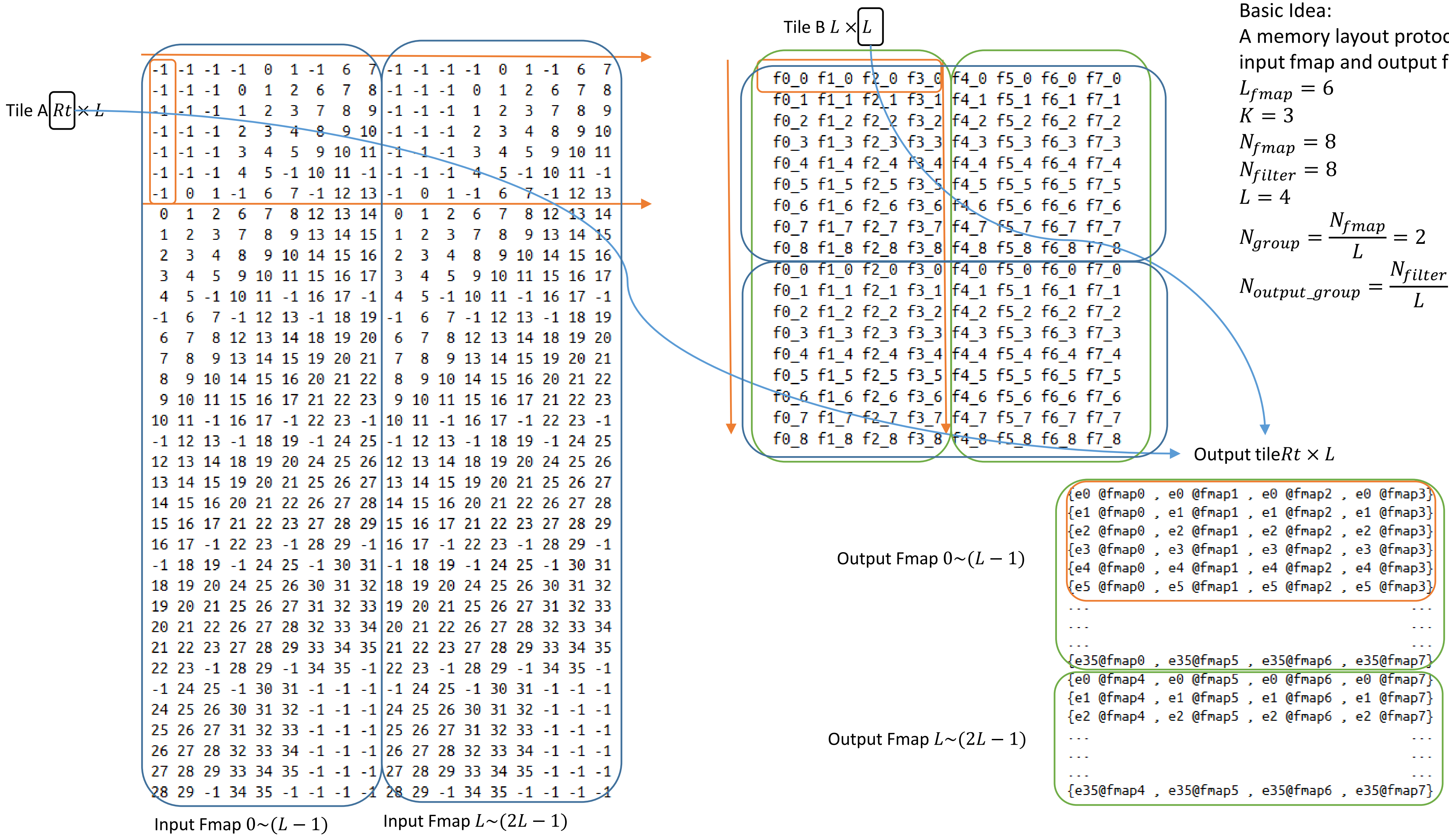
$N_{fmap} = 8$

$N_{filter} = 8$

$L = 4$

$$N_{group} = \frac{N_{fmap}}{L} = 2$$

$$N_{output\_group} = \frac{N_{filter}}{L} = 2$$



-1	-1	-1	-1	0	1	-1	6	7	-1	-1	-1	-1	0	1	-1	6	7
-1	-1	-1	0	1	2	6	7	8	-1	-1	-1	0	1	2	6	7	8
-1	-1	-1	1	2	3	7	8	9	-1	-1	-1	1	2	3	7	8	9
-1	-1	-1	2	3	4	8	9	10	-1	-1	-1	2	3	4	8	9	10
-1	-1	-1	3	4	5	9	10	11	-1	-1	-1	3	4	5	9	10	11
-1	-1	-1	4	5	-1	10	11	-1	-1	-1	-1	4	5	-1	10	11	-1
-1	0	1	-1	6	7	-1	12	13	-1	0	1	-1	6	7	-1	12	13
0	1	2	6	7	8	12	13	14	0	1	2	6	7	8	12	13	14
1	2	3	7	8	9	13	14	15	1	2	3	7	8	9	13	14	15
2	3	4	8	9	10	14	15	16	2	3	4	8	9	10	14	15	16
3	4	5	9	10	11	15	16	17	3	4	5	9	10	11	15	16	17
4	5	-1	10	11	-1	16	17	-1	4	5	-1	10	11	-1	16	17	-1
-1	6	7	-1	12	13	-1	18	19	-1	6	7	-1	12	13	-1	18	19
6	7	8	12	13	14	18	19	20	6	7	8	12	13	14	18	19	20
7	8	9	13	14	15	19	20	21	7	8	9	13	14	15	19	20	21
8	9	10	14	15	16	20	21	22	8	9	10	14	15	16	20	21	22
9	10	11	15	16	17	21	22	23	9	10	11	15	16	17	21	22	23
10	11	-1	16	17	-1	22	23	-1	10	11	-1	16	17	-1	22	23	-1
-1	12	13	-1	18	19	-1	24	25	-1	12	13	-1	18	19	-1	24	25
12	13	14	18	19	20	24	25	26	12	13	14	18	19	20	24	25	26
13	14	15	19	20	21	25	26	27	13	14	15	19	20	21	25	26	27
14	15	16	20	21	22	26	27	28	14	15	16	20	21	22	26	27	28
15	16	17	21	22	23	27	28	29	15	16	17	21	22	23	27	28	29
16	17	-1	22	23	-1	28	29	-1	16	17	-1	22	23	-1	28	29	-1
-1	18	19	-1	24	25	-1	30	31	-1	18	19	-1	24	25	-1	30	31
18	19	20	24	25	26	30	31	32	18	19	20	24	25	26	30	31	32
19	20	21	25	26	27	31	32	33	19	20	21	25	26	27	31	32	33
20	21	22	26	27	28	32	33	34	20	21	22	26	27	28	32	33	34
21	22	23	27	28	29	33	34	35	21	22	23	27	28	29	33	34	35
22	23	-1	28	29	-1	34	35	-1	22	23	-1	28	29	-1	34	35	-1
-1	24	25	-1	30	31	-1	-1	-1	-1	24	25	-1	30	31	-1	-1	-1
24	25	26	30	31	32	-1	-1	-1	24	25	26	30	31	32	-1	-1	-1
25	26	27	31	32	33	-1	-1	-1	25	26	27	31	32	33	-1	-1	-1
26	27	28	32	33	34	-1	-1	-1	26	27	28	32	33	34	-1	-1	-1
27	28	29	33	34	35	-1	-1	-1	27	28	29	33	34	35	-1	-1	-1
28	29	-1	34	35	-1	-1	-1	-1	28	29	-1	34	35	-1	-1	-1	-1

```
f0_0 f1_0 f2_0 f3_0 f4_0 f5_0 f6_0 f7_0
f0_1 f1_1 f2_1 f3_1 f4_1 f5_1 f6_1 f7_1
f0_2 f1_2 f2_2 f3_2 f4_2 f5_2 f6_2 f7_2
f0_3 f1_3 f2_3 f3_3 f4_3 f5_3 f6_3 f7_3
f0_4 f1_4 f2_4 f3_4 f4_4 f5_4 f6_4 f7_4
f0_5 f1_5 f2_5 f3_5 f4_5 f5_5 f6_5 f7_5
f0_6 f1_6 f2_6 f3_6 f4_6 f5_6 f6_6 f7_6
f0_7 f1_7 f2_7 f3_7 f4_7 f5_7 f6_7 f7_7
f0_8 f1_8 f2_8 f3_8 f4_8 f5_8 f6_8 f7_8
f0_0 f1_0 f2_0 f3_0 f4_0 f5_0 f6_0 f7_0
f0_1 f1_1 f2_1 f3_1 f4_1 f5_1 f6_1 f7_1
f0_2 f1_2 f2_2 f3_2 f4_2 f5_2 f6_2 f7_2
f0_3 f1_3 f2_3 f3_3 f4_3 f5_3 f6_3 f7_3
f0_4 f1_4 f2_4 f3_4 f4_4 f5_4 f6_4 f7_4
f0_5 f1_5 f2_5 f3_5 f4_5 f5_5 f6_5 f7_5
f0_6 f1_6 f2_6 f3_6 f4_6 f5_6 f6_6 f7_6
f0_7 f1_7 f2_7 f3_7 f4_7 f5_7 f6_7 f7_7
f0_8 f1_8 f2_8 f3_8 f4_8 f5_8 f6_8 f7_8
```

```
{e0 @fmap0 , e0 @fmap1 , e0 @fmap2 , e0 @fmap3}
{e1 @fmap0 , e1 @fmap1 , e1 @fmap2 , e1 @fmap3}
{e2 @fmap0 , e2 @fmap1 , e2 @fmap2 , e2 @fmap3}
{e3 @fmap0 , e3 @fmap1 , e3 @fmap2 , e3 @fmap3}
{e4 @fmap0 , e4 @fmap1 , e4 @fmap2 , e4 @fmap3}
{e5 @fmap0 , e5 @fmap1 , e5 @fmap2 , e5 @fmap3}
...                                          ...
...                                          ...
...                                          ...
{e35@fmap0 , e35@fmap5 , e35@fmap6 , e35@fmap7}
{e0 @fmap4 , e0 @fmap5 , e0 @fmap6 , e0 @fmap7}
{e1 @fmap4 , e1 @fmap5 , e1 @fmap6 , e1 @fmap7}
{e2 @fmap4 , e2 @fmap5 , e2 @fmap6 , e2 @fmap7}
...                                          ...
...                                          ...
...                                          ...
{e35@fmap4 , e35@fmap5 , e35@fmap6 , e35@fmap7}
```

Target Matrix A

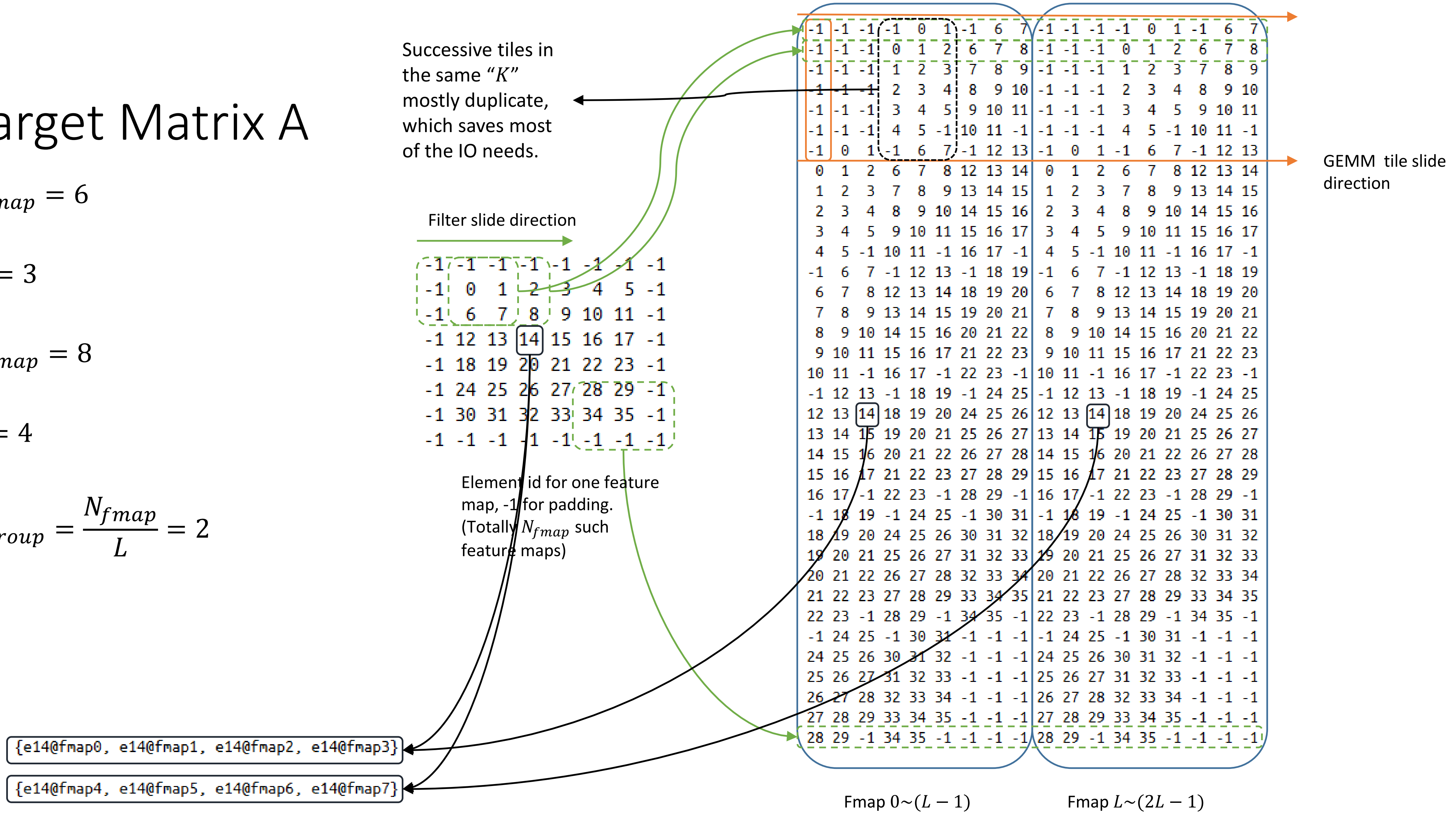
$L_{fmap} = 6$

$K = 3$

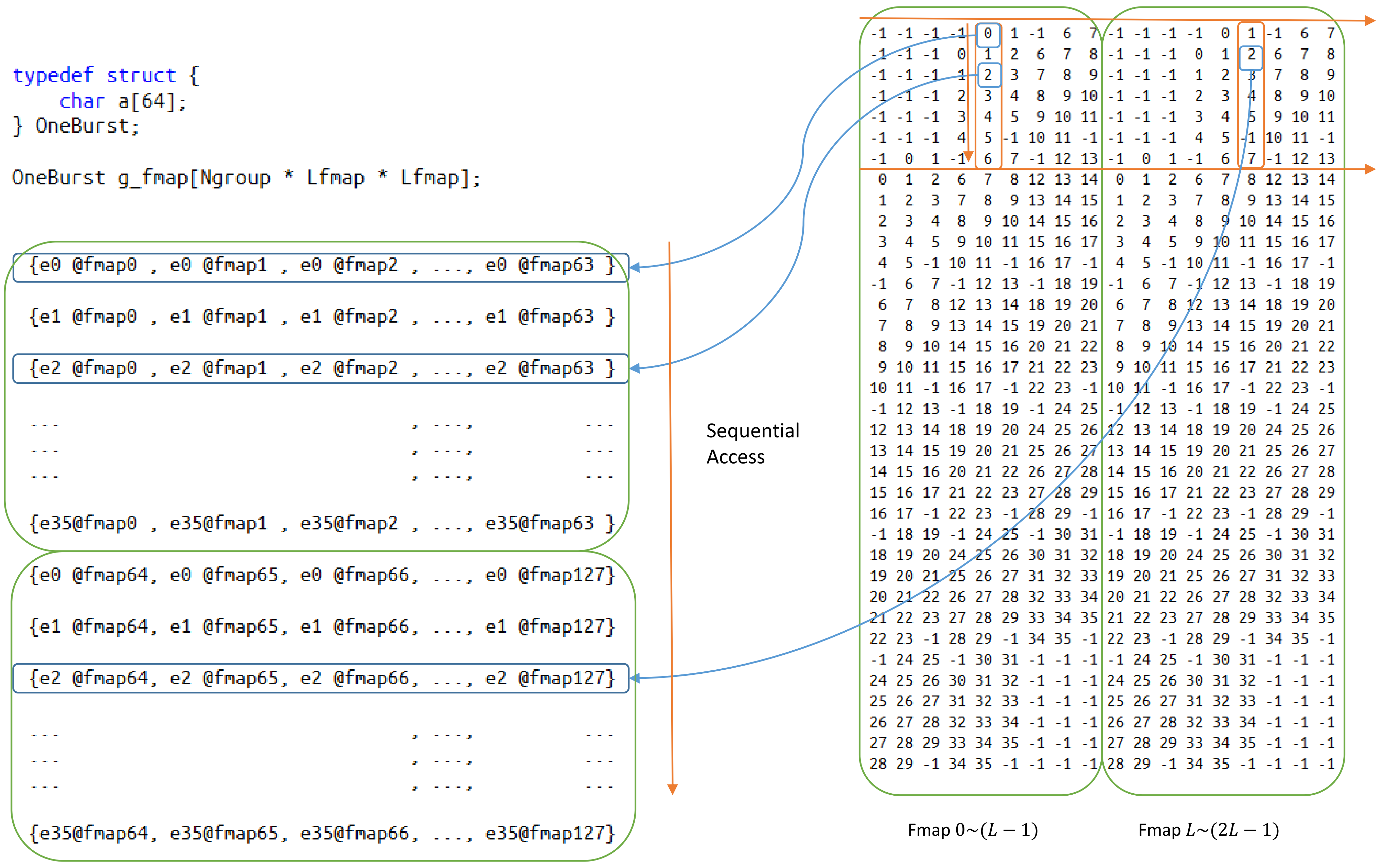
$N_{fmap} = 8$

$L = 4$

$N_{group} = \frac{N_{fmap}}{L} = 2$



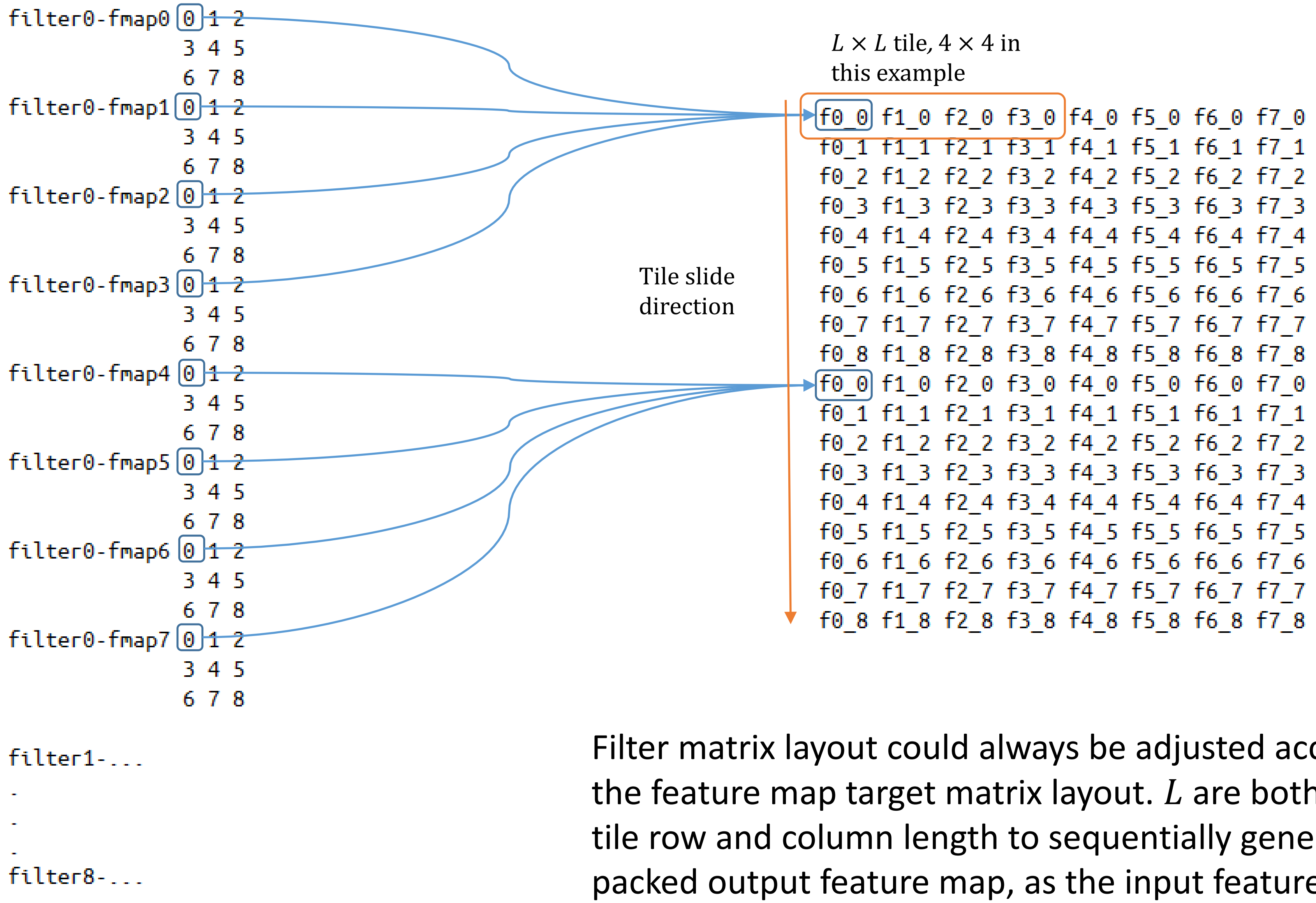
Successive tiles in the same "K" mostly duplicate, which saves most of the IO needs.
Filter slide direction
Element id for one feature map, -1 for padding. (Totally N_{fmap} such feature maps)
GEMM tile slide direction
Fmap $0\sim(L-1)$
Fmap $L\sim(2L-1)$
{e14@fmap0, e14@fmap1, e14@fmap2, e14@fmap3}
{e14@fmap4, e14@fmap5, e14@fmap6, e14@fmap7}



```
typedef struct {
    char a[64];
} OneBurst;

OneBurst g_fmap[Ngroup * Lfmap * Lfmap];
```

```
{e0 @fmap0 , e0 @fmap1 , e0 @fmap2 , ..., e0 @fmap63 }

{e1 @fmap0 , e1 @fmap1 , e1 @fmap2 , ..., e1 @fmap63 }

{e2 @fmap0 , e2 @fmap1 , e2 @fmap2 , ..., e2 @fmap63 }

...                                   , ...,          ...
...                                   , ...,          ...
...                                   , ...,          ...

{e35@fmap0 , e35@fmap1 , e35@fmap2 , ..., e35@fmap63 }
{e0 @fmap64, e0 @fmap65, e0 @fmap66, ..., e0 @fmap127}

{e1 @fmap64, e1 @fmap65, e1 @fmap66, ..., e1 @fmap127}

{e2 @fmap64, e2 @fmap65, e2 @fmap66, ..., e2 @fmap127}

...                                   , ...,          ...
...                                   , ...,          ...
...                                   , ...,          ...

{e35@fmap64, e35@fmap65, e35@fmap66, ..., e35@fmap127}
```

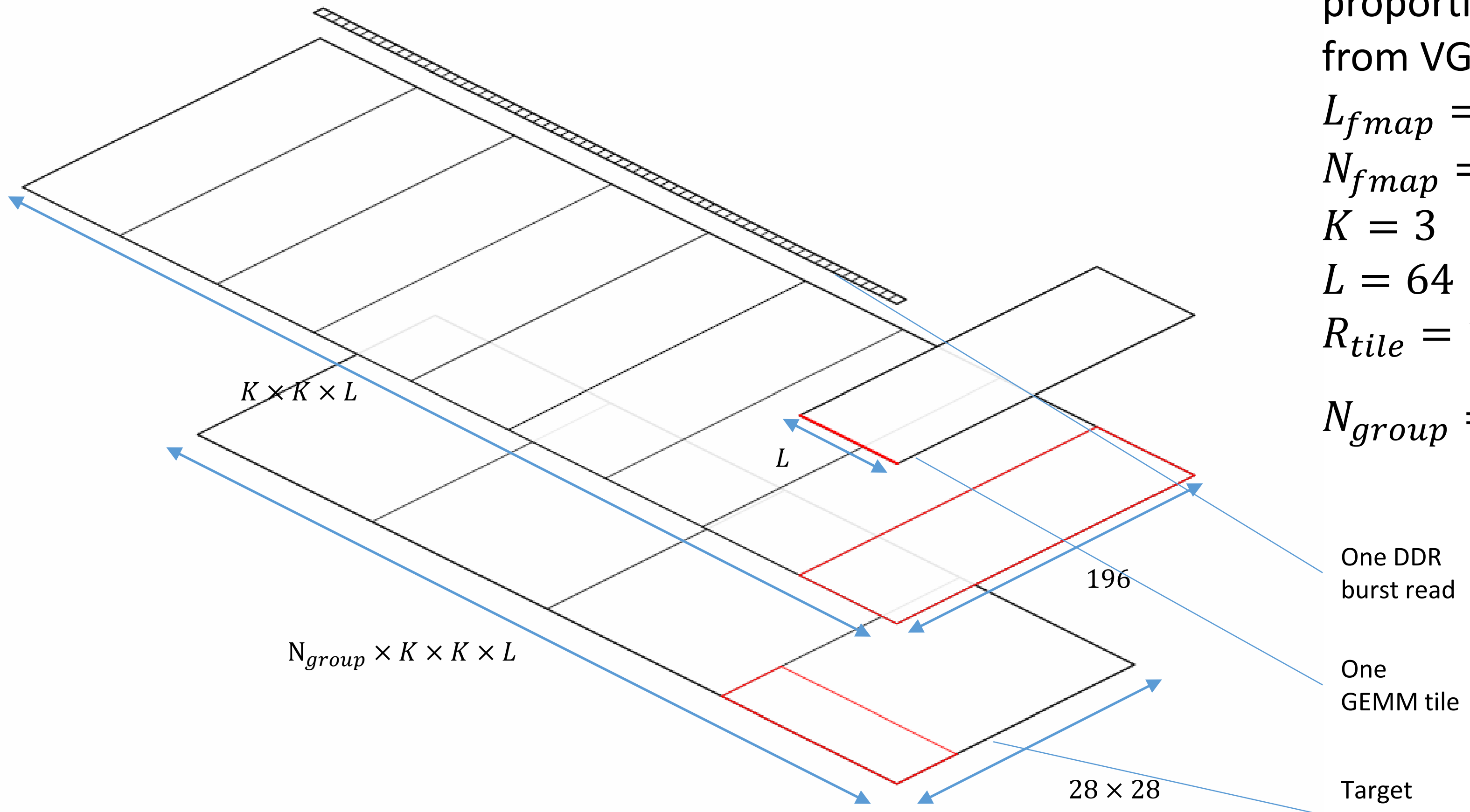
```
-1 -1 -1 -1  0  1 -1  6  7 -1 -1 -1 -1  0  1 -1  6  7
-1 -1 -1  0  1  2  6  7  8 -1 -1 -1  0  1  2  6  7  8
-1 -1 -1  1  2  3  7  8  9 -1 -1 -1  1  2  3  7  8  9
-1 -1 -1  2  3  4  8  9 10 -1 -1 -1  2  3  4  8  9 10
-1 -1 -1  3  4  5  9 10 11 -1 -1 -1  3  4  5  9 10 11
-1 -1 -1  4  5 -1 10 11 -1 -1 -1 -1  4  5 -1 10 11 -1
-1  0  1 -1  6  7 -1 12 13 -1  0  1 -1  6  7 -1 12 13
 0  1  2  6  7  8 12 13 14  0  1  2  6  7  8 12 13 14
 1  2  3  7  8  9 13 14 15  1  2  3  7  8  9 13 14 15
 2  3  4  8  9 10 14 15 16  2  3  4  8  9 10 14 15 16
 3  4  5  9 10 11 15 16 17  3  4  5  9 10 11 15 16 17
 4  5 -1 10 11 -1 16 17 -1  4  5 -1 10 11 -1 16 17 -1
-1  6  7 -1 12 13 -1 18 19 -1  6  7 -1 12 13 -1 18 19
 6  7  8 12 13 14 18 19 20  6  7  8 12 13 14 18 19 20
 7  8  9 13 14 15 19 20 21  7  8  9 13 14 15 19 20 21
 8  9 10 14 15 16 20 21 22  8  9 10 14 15 16 20 21 22
 9 10 11 15 16 17 21 22 23  9 10 11 15 16 17 21 22 23
10 11 -1 16 17 -1 22 23 -1 10 11 -1 16 17 -1 22 23 -1
-1 12 13 -1 18 19 -1 24 25 -1 12 13 -1 18 19 -1 24 25
12 13 14 18 19 20 24 25 26 12 13 14 18 19 20 24 25 26
13 14 15 19 20 21 25 26 27 13 14 15 19 20 21 25 26 27
14 15 16 20 21 22 26 27 28 14 15 16 20 21 22 26 27 28
15 16 17 21 22 23 27 28 29 15 16 17 21 22 23 27 28 29
16 17 -1 22 23 -1 28 29 -1 16 17 -1 22 23 -1 28 29 -1
-1 18 19 -1 24 25 -1 30 31 -1 18 19 -1 24 25 -1 30 31
18 19 20 24 25 26 30 31 32 18 19 20 24 25 26 30 31 32
19 20 21 25 26 27 31 32 33 19 20 21 25 26 27 31 32 33
20 21 22 26 27 28 32 33 34 20 21 22 26 27 28 32 33 34
21 22 23 27 28 29 33 34 35 21 22 23 27 28 29 33 34 35
22 23 -1 28 29 -1 34 35 -1 22 23 -1 28 29 -1 34 35 -1
-1 24 25 -1 30 31 -1 -1 -1 -1 24 25 -1 30 31 -1 -1 -1
24 25 26 30 31 32 -1 -1 -1 24 25 26 30 31 32 -1 -1 -1
25 26 27 31 32 33 -1 -1 -1 25 26 27 31 32 33 -1 -1 -1
26 27 28 32 33 34 -1 -1 -1 26 27 28 32 33 34 -1 -1 -1
27 28 29 33 34 35 -1 -1 -1 27 28 29 33 34 35 -1 -1 -1
28 29 -1 34 35 -1 -1 -1 -1 28 29 -1 34 35 -1 -1 -1 -1
```



```
filter0-fmap0 0 1 2
              3 4 5
              6 7 8
filter0-fmap1 0 1 2
              3 4 5
              6 7 8
filter0-fmap2 0 1 2
              3 4 5
              6 7 8
filter0-fmap3 0 1 2
              3 4 5
              6 7 8
filter0-fmap4 0 1 2
              3 4 5
              6 7 8
filter0-fmap5 0 1 2
              3 4 5
              6 7 8
filter0-fmap6 0 1 2
              3 4 5
              6 7 8
filter0-fmap7 0 1 2
              3 4 5
              6 7 8

filter1-...
-
-
-
filter8-...
```

```
f0_0 f1_0 f2_0 f3_0 f4_0 f5_0 f6_0 f7_0
f0_1 f1_1 f2_1 f3_1 f4_1 f5_1 f6_1 f7_1
f0_2 f1_2 f2_2 f3_2 f4_2 f5_2 f6_2 f7_2
f0_3 f1_3 f2_3 f3_3 f4_3 f5_3 f6_3 f7_3
f0_4 f1_4 f2_4 f3_4 f4_4 f5_4 f6_4 f7_4
f0_5 f1_5 f2_5 f3_5 f4_5 f5_5 f6_5 f7_5
f0_6 f1_6 f2_6 f3_6 f4_6 f5_6 f6_6 f7_6
f0_7 f1_7 f2_7 f3_7 f4_7 f5_7 f6_7 f7_7
f0_8 f1_8 f2_8 f3_8 f4_8 f5_8 f6_8 f7_8
f0_0 f1_0 f2_0 f3_0 f4_0 f5_0 f6_0 f7_0
f0_1 f1_1 f2_1 f3_1 f4_1 f5_1 f6_1 f7_1
f0_2 f1_2 f2_2 f3_2 f4_2 f5_2 f6_2 f7_2
f0_3 f1_3 f2_3 f3_3 f4_3 f5_3 f6_3 f7_3
f0_4 f1_4 f2_4 f3_4 f4_4 f5_4 f6_4 f7_4
f0_5 f1_5 f2_5 f3_5 f4_5 f5_5 f6_5 f7_5
f0_6 f1_6 f2_6 f3_6 f4_6 f5_6 f6_6 f7_6
f0_7 f1_7 f2_7 f3_7 f4_7 f5_7 f6_7 f7_7
f0_8 f1_8 f2_8 f3_8 f4_8 f5_8 f6_8 f7_8
```

Filter matrix layout could always be adjusted according to the feature map target matrix layout. L are both used for tile row and column length to sequentially generate L-packed output feature map, as the input feature map for the next layer.



Target matrix at real proportion (One layer from VGG):

$L_{fmap} = 28$

$N_{fmap} = 256$

$K = 3$

$L = 64$

$R_{tile} = 196$

$$N_{group} = \frac{N_{fmap}}{L} = 4$$

Benefit for the optimized CNN memory layout

- **All DDR read inside one tile are sequential**, 64B/cycle V.S. 1B/(multiple cycles).
- **All DDR write at GEMM output are sequential.**
- **No interlayer convert needed.** Input and output feature map conform the same memory layout, layer N – 1’s output could be seamlessly used for the next layer.
- **Consecutive tile content largely overlap.** (0, 1, 2, 3, …) V.S. (1, 2, 3, 4, …), could be reused, saves a big part of the DDR access, reuses every fetched number K times.
- **Easy to generalize.** This format requires Nfmap % L == 0, which could be easily achieved by padding extra all-zero feature maps.

Experiment result

Model: VGG19 [1]

Board: Microsoft Catapult FPGA acceleration card, version: Pikespeak

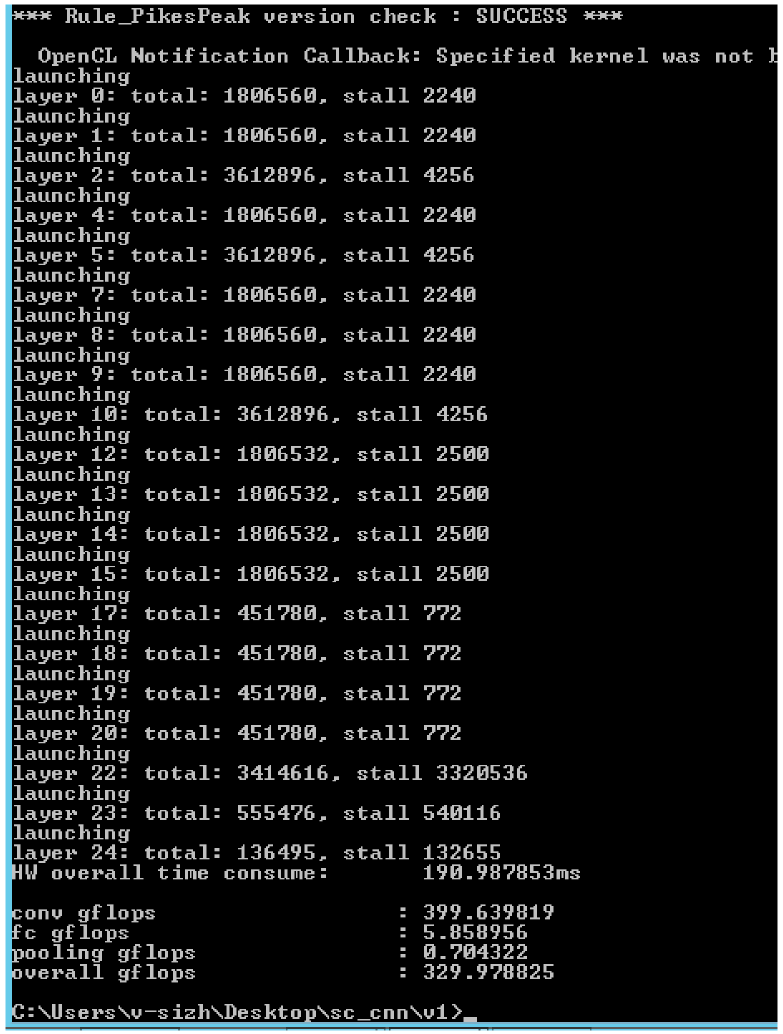
Precision: 8bit fix number

Overall perf: 329.9 gops

Convolution only perf: 399.6 gops

GEMM theoretical limit: 200MHz X 1024DSP X 2 = 409.6 gops

Note: Fully connect layers applies $M \times V$, performance of this operation is limited by DDR bandwidth, as could be seen from the screenshot, in the last three fully connect layers, stalled cycles(calculation waiting for IO) takes most of the total execution cycles.



```
*** Rule_PikesPeak version check : SUCCESS ***

  OpenCL Notification Callback: Specified kernel was not b
launching
layer 0: total: 1806560, stall 2240
launching
layer 1: total: 1806560, stall 2240
launching
layer 2: total: 3612896, stall 4256
launching
layer 4: total: 1806560, stall 2240
launching
layer 5: total: 3612896, stall 4256
launching
layer 7: total: 1806560, stall 2240
launching
layer 8: total: 1806560, stall 2240
launching
layer 9: total: 1806560, stall 2240
launching
layer 10: total: 3612896, stall 4256
launching
layer 12: total: 1806532, stall 2500
launching
layer 13: total: 1806532, stall 2500
launching
layer 14: total: 1806532, stall 2500
launching
layer 15: total: 1806532, stall 2500
launching
layer 17: total: 451780, stall 772
launching
layer 18: total: 451780, stall 772
launching
layer 19: total: 451780, stall 772
launching
layer 20: total: 451780, stall 772
launching
layer 22: total: 3414616, stall 3320536
launching
layer 23: total: 555476, stall 540116
launching
layer 24: total: 136495, stall 132655
HW overall time consume:       190.987853ms

conv gflops                 : 399.639819
fc gflops                   : 5.858956
pooling gflops              : 0.704322
overall gflops              : 329.978825

C:\Users\v-sizh\Desktop\sc_cnn\v1>_
```

[1] Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge."*International Journal of Computer Vision* 115.3 (2015): 211-252.

Comparison to related works

	[1] ICCD'13	[2] FPGA'15	[3] FPGA'16	[3] FPGA'16	this work
FPGA	Virtex-6 VLX240T	Virtex-7 VX485T	Stratix-V GSD8	Stratix-V GSD8	Stratix-V GSMD5
Frequency	150 MHz	100 MHz	140 MHz	120 MHz	200 MHz
CNN model	2.74 GMAC	AlexNet 1.33 GOP	AlexNet 1.46 GOP	VGG-16 30.9 GOP	VGG-19
Precision	fixed	float (32b)	fixed (8-16b)	fixed (8-16b)	Fixed (8b)
Performance	17 GOPS[b]	61.6 GOPS[a]	126.6 GOPS[a] 72.4 GOPS[b]	136.5 GOPS[a] 117.8 GOPS[b]	399.6 GOPSa 329.9 GOPSb

[a] convolution operations only [b] all operations for image classification

1 GMAC = 2 GOP

[1] M. Peemen, et al. Memory-centric accelerator design for convolutional neural networks. In *ICCD* 2013.

[2] C. Zhang, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *ACM ISFPGA* 2015.

[3] Suda, Naveen, et al. "Throughput-Optimized OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks." *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016.

Thanks