STACKED FILTERS STATIONARY FLOW FOR HARDWARE-ORIENTED ACCELERATION OF DEEP CONVOLUTIONAL NEURAL NETWORKS

Gao Yuechao, Liu Nianhong & Zhang Sheng *
Department of Microelectrics and Nanoelectrics
Tsinghua University
Beijing, 100084, China
{gyc15,lnh15}@mails.tsinghua.edu.cn

ABSTRACT

To address memory and computation resource limitations for hardware-oriented acceleration of deep convolutional neural networks (CNNs), we present a computation flow, stacked filters stationary flow (SFS), and a corresponding data encoding format, relative indexed compressed sparse filter format (CSF), to make the best of data sparsity, and simplify data handling at execution time. Comparing with the state-of-the-art result (Han et al., 2016b), just applying a few simple post-processing steps, our method achieved 1.11× improvement in reducing the storage required by AlexNet, and 1.09× improvement in reducing the storage required by SqueezeNet, without loss of accuracy on the ImageNet dataset. Moreover, using this approach, chip area for logics handling irregular sparse data access can be saved (about 19.1% chip area in (Han et al., 2016a)). Comparing with the 2D-SIMD processure structures in DVAS, ENVISION, etc., our method achieved about 3.65× computation efficiency improvement, from 26.4% to 96.5%, using the data from Deep Compression on AlexNet[1].

1 INTRODUCTION

CNNs have achieved substantial progress during the past years. But hardware resource limitations have hindered their wide usage in embedded devices. Various efforts have been made to address this issue, such as ShiftCNN (Gudovskiy & Rigazio, 2017), Ristretto (Gysel, 2016), Eyeriss (Chen et al., 2017), Deep Compression (Han et al., 2016b) and EIE (Han et al., 2016a). Through compressing deep neural networks with pruning, trained quantization and Huffman coding, Deep Compression (Han et al., 2016b), the best paper of ICLR 2016, achieved state-of-the-art result in reducing storage requirement of neural networks without affecting their accuracy.

In spite of the great progress achieved till now, there are still many problems to be solved. The first problem is manipulating compressed sparse data consumes considerable extra clock cycles and need extra logics (about 19.1% chip area in (Han et al., 2016a)). Several sparse matrix encoding formats have been proposed, such as CSC, CSR and CISR (Fowers et al., 2014). But existing encoding formats complex the computation at runtime due to its irregular memory access characteristics. This results in inefficiency in parallelizing computation and bigger chip area. Therefore, it would be desirable if the encoding can be directly used during run time. The second one is, for deeply compressed sparse networks, the computation efficiency of recently proposed hardware acceleration designs, such as Eyeriss (Chen et al., 2017), DVAS (Moons & Verhelst, 2015), ENVISION (Moons et al., 2017), DNPU (Shin et al., 2017), etc. , is fairly low. In this paper we present a novel computation flow SFS, and a corresponding memory layout and encoding format CSF which achieves the desirable goal that it can be directly used at run time. We also propose a three dimensional Single Instruction Multiple Data (3D-SIMD) processor architecture to illustrate how to accelerate deep CNNs by taking advantage of SFS flow and CSF format.

*Corresponding author. zhang_sh@tsinghua.edu.cn

[1]https://github.com/songhan/Deep-Compression-AlexNet

2 STACKED FILTERS STATIONARY FLOW (SFS) AND RELATIVE INDEXED COMPRESSED SPARSE FILTER (CSF) FORMAT

Computations of convolutional and fully connected layers in CNNs can be unified into one formula Eq.1 (ignoring biases). Eq. 2-6 illustrate our approach SFS. $\boldsymbol{V}_o$, $\boldsymbol{V}_i$ and $\boldsymbol{W}_f$ are the matrices of output feature maps, input feature maps and filters, respectively. S, C, K, M, M', m is a given stride size, channel number, filter kernel size, total filter number, number of batches and batch size. Filters are firstly grouped into M' batches with batch size m, and each ${\boldsymbol{W}_f}^{(n)}$ is then reshaped to ${\boldsymbol{W}_{f'}}^{(n)}$. One channel of feature data will convolute with m filters from the same channel in parallel (Eq.4, $j = 0, ..., m-1$, as illustrated in figure 2). At the end of computation, ${\boldsymbol{V}_{o'}}^{(n)}$ should be concatenated back to $\boldsymbol{V}_o$.

$$\boldsymbol{V}_o[cho][y][x] = \sum_{chi=0}^{C-1} \sum_{r=0}^{K-1} \sum_{c=0}^{K-1} \boldsymbol{W}_f[cho][chi][r][c] \times \boldsymbol{V}_i[chi][Sy+r][Sx+c] \tag{1}$$

$$\boldsymbol{W}_f = [{\boldsymbol{W}_f}^{(0)}, {\boldsymbol{W}_f}^{(1)}, ..., {\boldsymbol{W}_f}^{(M'-1)}], \boldsymbol{W}_{f'} = [{\boldsymbol{W}_{f'}}^{(0)}, {\boldsymbol{W}_{f'}}^{(1)}, ..., {\boldsymbol{W}_{f'}}^{(M'-1)}] \tag{2}$$

$${\boldsymbol{W}_{f'}}^{(n)}[chi][r][c][j] = {\boldsymbol{W}_f}^{(n)}[j][chi][r][c] \tag{3}$$

$${\boldsymbol{V}_{o'}}^{(n)}[j][y][x] = \sum_{chi=0}^{C-1} \sum_{r=0}^{K-1} \sum_{c=0}^{K-1} {\boldsymbol{W}_{f'}}^{(n)}[chi][r][c][j] \times \boldsymbol{V}_i[chi][Sy+r][Sx+c] \tag{4}$$

$$\boldsymbol{V}_o = [{\boldsymbol{V}_{o'}}^{(0)}, {\boldsymbol{V}_{o'}}^{(1)}, ..., {\boldsymbol{V}_{o'}}^{(M'-1)}] \tag{5}$$

$$M' = M/m, 0 \leq n < M', 0 \leq j < m, 0 \leq cho < M. \tag{6}$$

As to the encoding format CSF, our approach is to further rearrange the memory layout of the m filters illustrated in figure 1-a, storing the elements column by column. So in computation flow SFS, when each element in the feature map multiplies with a column of data from m filters, the filter weights could be loaded sequentially. The first line in figure 1-b illustrates the changing. If there is any weight value equals to 0, just remove the column contains that value, add one to the relative index in the next column, and subtract one to the pointer next to it. The non-zero value number (includes padding zeros) of a column is given by the next pointer. Column pointer is 0 means all the values in the column before the pointer equal to 0. Relative column pointer is not needed when parameters are stored in files.

Filter 1	W1,11	W1,12	W1,13	W1,21	...	W1,33
...	...	...	...	...	...	...
Filter m	Wm,11	Wm,12	Wm,13	Wm,21	...	Wm,33

a)

Virtual weight value	W1,11	W2,11	...	Wm,11	...	W1,kk	W2,kk	...	Wm,kk
Relative filter index	0	0	...	0	...	0	0	...	0
Relative column pointer	0	m	...	m	m				

b)

Figure 1: Memory layout a) for m filters with kernel size 3 from a single channel. b) in CSF format.

3 3D-SIMD PROCESSOR ARCHITECTURE

The SFS flow and the CSF encoding format are the key features of the proposed 3D-SIMD processor architecture, see figure 3. In this architecture, after feature data are loaded into the line buffer and

```
for chi in [0,C−1] {
  for r in [0,K−1] {
    for c in [0,K−1] {
      output channel 0: V_o^(n)[0][y][x]+=W_f^(n)[chi][r][c][0]×V_i[chi][Sy+r][Sx+c]
      output channel 1: V_o^(n)[1][y][x]+=W_f^(n)[chi][r][c][1]×V_i[chi][Sy+r][Sx+c]
                                    ...
      output channel m−1: V_o^(n)[m−1][y][x]+=W_f^(n)[chi][r][c][m−1]×V_i[chi][Sy+r][Sx+c]
    }
  }
}
```

Figure 2: SFS convolution pseudo code.

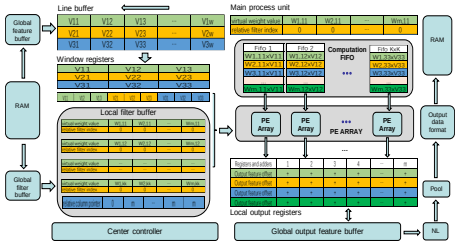

Figure 3: 3D-SIMD processor architecture.

Table 1: Extra space(in bits) improvement and total storage requirement improvement

Network	Nonzeros	Extra space	Improvement	Total
AlexNet by (Han et al., 2016b)	30592056	41325956		
AlexNet by SFS+CSF	30592056	34138970	1.21×	1.11×
SqueezeNet by (Han et al., 2016b)	3327368	1737628		
SqueezeNet by SFS+CSF	3327368	1307160	1.33×	1.09×

window registers from a single channel of input feature map, and m filter data from the same channel are loaded into the local buffer, each element in the window will multiply with a column of data from m filters at the same position. So data in CSF format can be directly used at run time, and zeros are skipped as designed. This demonstration shows that our approach can greatly simplify sparse data handling, saving zero bypassing and data looking up time. There is no complex sparse data handling logic is needed comparing with former works(Moons & Verhelst, 2015; Moons et al., 2017; Han et al., 2016a).

4 RESULT

The distribution of continuous zero numbers after applying the changing is first evaluated. As figure 4 shows, the distribution narrows to the left. It means that fewer bits is needed to store the relative index values, and there will be fewer padding zeros when compressing data in encoding formats. This will further reduce storage space. The distribution of continuous nonzero numbers is also evaluated. It also narrows to the left, which means the computation load during execution will be better balanced comparing to the reference work. The effect of batch size m is also analyzed. It shows that there do exist an optimum batch size for each layer. Smaller batch size requires smaller local buffer, but the input feature map need to be loaded more times. For simplicity, all the experiments in this section use filter number as the batch size. That means, all the filters and input feature maps are loaded only one time and the output feature maps are saved one time for one reference. Table 1 illustrate the improvement of extra space needed to store index and padding zeros, and the total improvement of storage requirement after applying our method comparing to former works. We also evaluated the computation efficiency improvement of convolutional and fully connected layers on several networks. On Alexnet, the number of MACs of dense network is 1.06GOPS, the number of MACs of non-zero weight value is 0.28GOPS, and the number of MACs after applying our method is 0.29GOPS. So comparing with dense network processor architectures, the computation efficiency improves from 26.4% to 96.5%, that is, about 3.65× improvement.

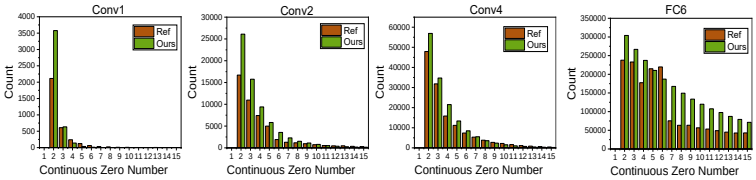

Figure 4: Distributions of continuous zero numbers in different layers of Alexnet, comparing with reference work (Han et al., 2016b).

5 CONCLUSION

In this paper, we propose a computation flow SFS, and its corresponding data encoding format CSF. Experimental results show that the characters of these two features can help to further compress the network parameters by about 8% to 10% and balance computation load at run time. By adopting the proposed 3D-SIMD processor architecture, chip area for logics handling irregular memory access of sparse data can be saved. Moreover, directly using the encoded data as it is at runtime can also save zero bypassing clock cycles, and data looking up time.

REFERENCES

Yu Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017.

Jeremy Fowers, Kalin Ovtcharov, Karin Strauss, Eric S. Chung, and Greg Stitt. A high memory bandwidth fpga accelerator for sparse matrix-vector multiplication. In *IEEE International Symposium on Field-Programmable Custom Computing Machines*, pp. 36–43, 2014.

Denis A Gudovskiy and Luca Rigazio. Shiftcnn: Generalized low-precision architecture for inference of convolutional neural networks. 2017.

Philipp Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. 2016.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *International Conference on Computer Architecture (ISCA)*, 2016a.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*, 2016b.

Bert Moons and Marian Verhelst. Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing. In *Ieee/acm International Symposium on Low Power Electronics and Design*, pp. 237–242, 2015.

Bert Moons, Roel Uytterhoeven, Wim Dehaene, and Marian Verhelst. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi. In *Solid-State Circuits Conference*, pp. 246–247, 2017.

Dongjoo Shin, Jinmook Lee, Jinsu Lee, and Hoi Jun Yoo. 14.2 dnpu: An 8.1tops/w reconfigurable cnn-rnn processor for general-purpose deep neural networks. In *Solid-State Circuits Conference*, pp. 240–241, 2017.