An FPGA Implementation of GCN with Sparse Adjacency Matrix

Luchang Ding[1], Zhize Huang[2], Gengsheng Chen[3]

[1 2 3] State Key Lab of ASIC and System, Fudan University, Shanghai 200433, China

Email: gschen@fudan.edu.cn

Abstract

Deep learning has revolutionized many machine learning tasks in recent years, ranging from image classification and video processing to speech recognition and natural language understanding. The data in these tasks are typically represented in the Euclidean space. However, there is an increasing number of applications where data are generated from non-Euclidean domains and are represented as graphs with complex relationships and interdependency between objects. The complexity of graph data has approaches for graph data have emerged.

Graph convolution networks (GCNs) in particular, try to replicate the success of CNN in graph data by defining graph convolutions via graph spectral theory or spatial locality. This paper presents a new GCN accelerator for the graph convolution layers of ST-GCN [1], which is successfully applied in action recognition. The accelerator breaks down the graph convolution into convolution and matrix multiplication with adjacency matrix. To optimize the power efficiency, the dataflow of the convolution is designed properly and a sparse matrix-vector multiplication is proposed to make use of the sparsity of the adjacency matrix. The accelerator is implemented on NSA.241 accelerator and can reach its peak performance of 46.0GOP/s under 188MHz with about 220 DSP, which achieves high DSP efficiency.

1. Introduction

In recent years, human motion recognition has become an active research field, which plays an important role in video understanding. Human behavior recognition has many modes, such as appearance, depth, light flow and body skeleton. In the form of 2D or 3D coordinates, dynamic skeletal modes can be naturally represented by time series of human joint positions. Then, human behavior recognition can be achieved by analyzing its action pattern. The early method of action recognition based on skeleton is to use joint coordinates to form feature vectors at each time step and to analyze their time series. However, these methods have limited capabilities because they do not explicitly utilize the spatial relationships of human joints, which are essential for understanding human behavior. Recently, researchers have developed new methods to try to use natural connections between joints. These improvements are encouraging and demonstrating the importance of bone connectivity.

Recently, the generalization of convolution neural network (CNN) to graph convolution network (GCN) with arbitrary structure graphics has attracted more and more attention. By extending graph convolution network to spatiotemporal graph model, a general representation of skeletal sequence for behavior recognition is designed, which is called spatiotemporal graph convolution network (ST-GCN). The model is developed on the skeleton map sequence, where each node corresponds to a joint of the human body.

The hierarchy of ST-GCN eliminates the need for manual partitioning or traversal rules. Using the spatial edge that corresponds to the joint's natural connection and the temporal edge that connects the same joint in successive time steps, not only can it achieve more expressive power and higher performance (as our experiments show), but it can also be easily promoted in different environments.

Despite the great success of GCN, few studies have focused on its hardware acceleration. Existing accelerators are mainly used for inference of CNN and RNN, and almost none of them have mentioned the acceleration of graph convolution. In this work, we explored the hardware acceleration potential of GCN by utilizing the sparsity of the graph convolution by using the graph convolution layers in ST-GCN as example.

2. Algorithm

2.1 ST-GCN Model

Bone-based data can be obtained from motion capture devices or video attitude estimation algorithms. Usually, the data is a series of frames, each frame has a set of joint coordinates. Given the sequence of body joints in 2D or 3D coordinates, we can construct a space-time map. Among them, the nodes of the corresponding graph of human joints, the connectivity of human body structure and the connectivity of time correspond to the two kinds of edges of the graph. Therefore, the input of ST-GCN is the joint coordinate vector of graph nodes. By applying multi-layer spatiotemporal convolution operation to input data, higher-level feature maps can be generated. Then, it will be classified into corresponding action categories by the standard SoftMax classifier.

In ST-GCN, space-time convolution map is used to form the expression of multi-layer skeletal sequences. The

978-1-7281-0735-6/19/$31.00 ©2019 IEEE

skeletal spatiotemporal map $G = (V, E)$ is constructed. The number of frames is T and the number of joints is N.

In this graph, the node set $\mathrm{V} = \{v_{t_i} \mid \mathrm{t} = 1,2,\ldots,\mathrm{T}, \mathrm{i} = 1,2,\ldots,\mathrm{N}\}$ includes all the joint points on the skeletal sequence. The skeletal sequence is constructed in two steps. In the first step, the edges between frames represent the temporal relationship of human skeleton points. In the second step, the spatial map is constructed according to the natural skeleton connection of human body within each frame. The establishment of such links relies on natural structures, and there is no manual design. This model also supports the use of different numbers of skeletal point sets. For example, Kinetics dataset has 18 joints while the number of joints in NTU-RGB+D dataset is 25.

$B(v_{ti}) = \{v_{tj} | d(v_{tj}, v_{ti}) \leq 1\}$ represents the 1-neighbor set of v_{ti}. ST-GCN designs a strategy to divide the neighbor set into $k_T = 3$ parts, so there are 3 adjacency matrix used in the graph convolution. The graph convolution applied in ST-GCN is equivalent to the cascade of 2D convolution and matrix multiplication with Adjacency matrix. Therefore, in our hardware implementation, we converted graph convolution into 2D convolution and matrix multiplication operations.

2.2 Sparse Matrix-Vector Representation

The adjacency matrix mentioned above is a sparse matrix. For this reason, we implemented sparse matrix-vector multiplication (SpMxV) in our hardware design. SpMxV outperforms dense matrix-vector multiplication in two aspects: (1) less storage space, (2) lower power consumption.

There are three main schemes to represent sparse matrix: coordinate (COO) format, compressed sparse row (CSR) format and the compressed sparse column (CSC) format. The main idea of such methods is to skip zero elements in the matrix. In this work, we adopted CSC, and the reason will be explained in section 3. Row indices, column pointers and nonzero values are stored in three different arrays in CSC, as shown in Formula (1).

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 5 & 8 & 0 & 0 \\ 0 & 0 & 2 & 3 \\ 0 & 9 & 0 & 0 \end{bmatrix}$$

$$CSC = \begin{cases} data = [5 \ \ 8 \ \ 9 \ \ 2 \ \ 3] \\ row = [1 \ \ 1 \ \ 3 \ \ 2 \ \ 2] \\ ptr = [0 \ \ 1 \ \ 3 \ \ 4 \ \ 5] \end{cases} \tag{1}$$

2.3 Quantization

To reduce the computational complexity of convolution and matrix multiplication, dynamic fixed point was used in our quantization scheme [2]. In dynamic fixed point, each number is represented as follows:

$$data = (-1)^s \cdot 2^{-fl} \cdot \sum_{i=0}^{B=2} 2^i \cdot x_i \tag{2}$$

Where s is the sign bit, fl is the fractional length, B is the total data width, which is 16 in our design, and x_i is the mantissa bit. Dynamic fixed point can eliminate the deficiency of fixed point in terms of representation capability. Dynamic fixed point is only slightly different with the fixed point in hardware implementation. Multiplication with 2^{-fl} is equivalent to shift operations.

3. Hardware Architecture

The system architecture of our design is demonstrated in Figure 1. Generally, the accelerator comprises four parts: Conv engine, SpMxV engine, post process engine and internal buffers. Conv engine is responsible for computing the convolution results which will be sent to SpMxV engine for matrix multiplication. Post process engine will apply ReLU and Batch Normalization to the data. Data reuse is explored with the help of internal buffers. Adja Buffer contains 3 parts: row indices buffer, column pointers buffer and nonzero values buffer. Module Config Regs is used to configure the parameter of the design, such as data address, fraction bits of data and so on.

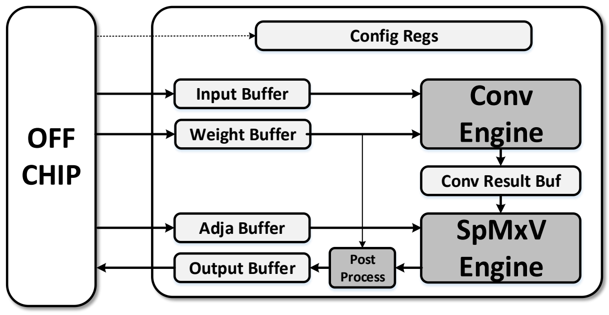

Figure 1. The system architecture

3.1 Conv Engine

Conv engine takes input and weights from input buffer and weight buffer and generate output to Conv result buffer. In order to facilitate the SpMxV engine to use the convolution result, the dataflow of Conv engine must be delicately designed. For the ambition of reusing the convolution result on-chip and reduce the size of output buffer, the Conv engine generate results row by row, and compute $k_T = 3$ row simultaneously. What's more, to further reduce the latency, the Conv engine parallel both the input channel dimension and the output channel dimension with the parallel factor $Tp = 8$.

The architecture of one Conv processing element inside the Conv engine is illustrated in Figure 2. In our design, we explore data reuse by weight stationary dataflow. Specifically, the weights will be reused until one row of partial sum is calculated. The MAC here is actually just a multiplier because the size of convolution kernel is 1. Input data is fed into PE by FIFOs and ping-pong buffer is applied to load weights in each PE in order to improve the throughput of the PE.

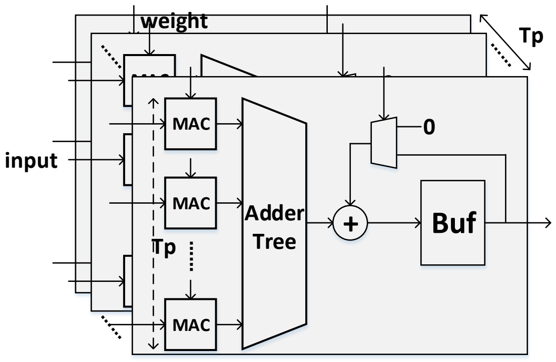

Figure 2. The architecture of Conv PE

3.2 SpMxV Engine

Each processing element in SpMxV engine takes one row vector x from Conv result buffer and an adjacency matrix A represented in CSC format as inputs. The engine decomposes matrix-vector multiplication into several inner product of two vectors. This process is expressed by Formula (3), where out_i is the value of the output vector with index i.

$$out_i = \sum_{j=0}^{V-1} x_j \cdot A_{ji} \tag{3}$$

The sparsity of adjacency matrix brings lots of benefits to our design, especially in the reduction of the number of MAC operations. When applying dense matrix-vector multiplication, the total times of MAC operation is expressed as the following formula, where V is the height and width of the adjacency matrix.

$$num_{mac} = V \cdot V \tag{4}$$

Suppose there are m non-zero values in the adjacency matrix, so the number of MAC operations needed in the SpMxV engine is m, noticing that in usual, $m \ll V \cdot V$. Taking the actual situation as example, the maximal number non-zero values in the adjacency matrix is 40 and $V = 25$. Hence, in our design, we can save at least 15x MAC operation, resulting gigantic power saving.

Moreover, by storing the row indices, column pointers and nonzero values instead of the matrix itself, CSC format is also an effective way to save storage space. The storage space occupied by these three arrays is m, $V + 1$ and m respectively and $2m + V + 1$ in total, compared to the number $V \cdot V$ when storing the whole adjacency matrix. As long as m is a small number, the storage spaces are effectively saved.

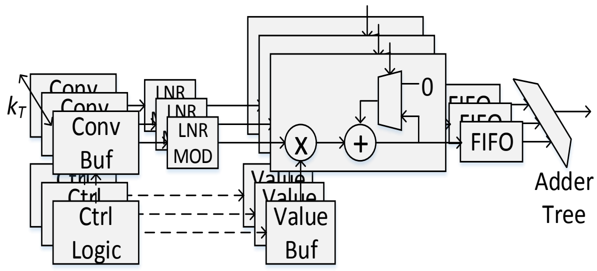

Figure 3. The architecture of SpMxV PE

Figure 3 displays the architecture of SpMxV PE. There are totally $Tp = 8$ PEs in the SpMxV engine. Ctrl logic in the PE generate the addresses used for reading data from Conv result buffer and non-zero value buffer according the row indices and the column pointers. LNR MOD is a module used for adding bias and doing overflow process. The FIFOs before the adder tree is necessary because the latency needed of each inner product depends on the sparsity of its adjacency matrix respectively. FIFOs are used to ensure the correctness of the generated values. The adder tree will only start to work when all FIFOs are not empty.

4. Experimental Result

In this section, we show our implementation result and evaluation result of the design. We implemented our design targeted on Xilinx UltraScale+™ FPGAs, XCVU9P, and the resource utilization of our design is shown in the following table.

Table 1 Resource Utilization

	Resource Utilization on XCVU9P			
	FF	LUT	DSP48	BRAM
Utilization	34667	44457	228	151
Available	2364480	1182240	6840	2160
Percent	1.5%	3.8%	3.3%	7.0%

We built an evaluation system as shown in Figure 4. The system was running at NSA.241 accelerator under 188MHz. The host PC can write data to or read data from the DDR4 on board. During evaluation, the Sparse GCN Engine first gets a start pulse, then reads the input data from the DDR4 and writes the output back to the DDR4, which can be accessed by the host PC.

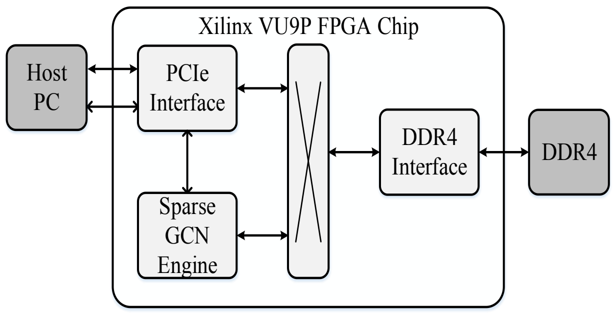

Figure 4 The evaluation system

Table 2 displays the latency of the accelerator when processing different graph convolution layers of the ST-GCN. The accelerator can process different layers according to different configurations.

Table 2 Performance on different layers (188MHz)

Input/Output Channels	Latency/ms
Layer1:3/64	2.61
Layer2:64/64	4.05
Layer3:64/128	8.03
Layer4:128/128	11.2
Layer5:128/256	22.3
Layer6:256/256	35.2

The accelerator achieves peak performance with 46.0GOP/s. when processing the Layer6. It should be noted that, when processing the layer1, the accelerator loses the ability to achieve its best performance since the input channel parallel factor is larger than input channel.

Table 3 DSP efficiency of our design

	FPGA 2016 [3]	ASICON2017 [4]	ASICON 2017 [5]	This work (peak)
Platform	Zynq XC7Z045	Xilinx VC709	ZCU102	NSA.241
Clock (MHz)	150	110	200	188
DSP Efficiency (GOP/s/DSP)	0.175	0.113	0.093	0.202

Table 3 shows that our design outperforms other works in terms of DSP efficiency. It also should be mentioned that our design has the potential to achieve better efficiency if ping-pong buffer is used between the Conv engine and SpMxV engine.

5. Conclusion and Discussion

Graph convolution has risen deep learning researchers' great interests in recent years, and it is becoming more and more popular. We preliminarily explored the architecture of graph convolution accelerator by utilizing the sparsity of adjacency matrix. In this work, we proposed a graph convolution accelerator for ST-GCN, which is a graph convolution neural network for skeleton-based action recognition. The peak performance of the accelerator can up to 46.0GOP/s with about 220 DSP.

Nonetheless, there are still lots of improvement can be done in the future. Firstly, we will further refine the structure of our accelerator in order to improve its performance. Secondly, we will map the entire ST-GCN to the FPGA in the future research.

References (follow the examples please)

[1] Yan S , Xiong Y , Lin D . Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition[J]. 2018.

[2] Gysel P , Motamedi M , Ghiasi S . Hardware-oriented Approximation of Convolutional Neural Networks[J]. 2016.

[3] Qiu J , Wang J , Yao S , et al. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network[C]// Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. ACM, 2016.

[4] Huang C , Ni S , Chen G . A layer-based structured design of CNN on FPGA[C]// 2017 IEEE 12th International Conference on ASIC (ASICON). IEEE, 2017.

[5] Li X , Cai Y , Han J , et al. A high utilization FPGA-based accelerator for variable-scale convolutional neural network[C]// 2017 IEEE 12th International Conference on ASIC (ASICON). IEEE, 2017.