FPGA Accelerating Core Design Based on XNOR Neural Network Algorithm

Su Yi[1,*], *Hu Xiao*[1], and Sun Yongjie[1]

[1] Graduate School of National of Defense Technology, Changsha, China

Abstract. The current deep learning application scenario is more and more extensive. In terms of computing platforms, the widely used GPU platforms have lower computational efficiency. The flexibility of APU-dedicated processors is difficult to deal with evolving algorithms, and the FPGA platform takes into account both computational flexibility and computational efficiency. At present, one of the bottlenecks for limiting large-scale deep learning algorithms on FPGA platforms is the large-scale floating-point computing. Therefore, this article studies single-bit parameterized quantized neural network algorithm (XNOR), and optimizes the neural network algorithm based on the structural characteristics of the FPGA platform. , Design and implementation of the FPGA acceleration core, the experimental results show that the acceleration effect is obvious.

1 INTRODUCTION

With the emergence of Alex Net in 2012, convolutional neural networks have been shown to be very effective for image processing. Deep learning using convolutional neural networks has gradually become the mainstream of image processing. The deep neural network overcomes the disadvantages of manual design and feature extraction in traditional image processing, which is time consuming, laborious, poor in robustness, and low in real-time running. Zhou Xingchen [1] used convolutional neural networks in handwritten Chinese character recognition. Liu [2] used CNN in automatic age and gender classification. Convolutional neural networks currently have many successful commercial image recognition applications, such as face recognition in public security by GTS Technology, object recognition in the automatic driving technology of Google Waymo and Tesla, and various types of Baidu and Alibaba. Among the cloud services for image search, deep neural networks have a large number of deployment requirements. The current GPU is still the platform that mainly runs these neural networks. It's powerful parallel computing structure controls neural network training and inference within a desired time range, and its flexible deployment capability can well adapt to the network iteration. However, its huge power consumption and relatively expensive price limit its deployment space to some extent. Although some research on the neural network training of FPGA, it still can't achieve the performance provided by GPU. However, there is an increasing demand of high-accuracy or real-time object detection tasks in large-scale clusters or embedded systems, which requires energy-efficient accelerators because of the green computation requirement or the limited battery restriction[3]. At present, a large number of studies have tended to deploy deep neural networks with low power consumption, and non-GPU hardware platforms have also become a research hotspot. This paper focuses on the inferential deployment of neural networks. There are two solutions that currently have the ability to challenge GPUs. One is ASIC, which has unparalleled performance advantages for a specific neural network. When the amount of use reaches a certain scale, the price will reach an optimal value. Its biggest drawback is that it cannot adapt well to the iteration of the network. The other platform is FPGA. FPGA platform also has low power consumption and has advantages in reconfigurability. At the stage when the current neural network model is not yet fully mature, it has significance to quickly implement the current existing neural network on the FPGA. Yu Zijian [4] conducted FPGA acceleration designs for handwritten digit recognition, and Nakahara [5], Li [6] designed low-precision weighted FPGA accelerators. Various types of algorithms for deep neural network optimization are currently being proposed, in which the compression weights and feature maps are input to ultra-low precision, so that the convolution operation includes only logical and integer operations, and the

* Corresponding author: totemxm@163.com

method is very suitable for FPGA implementation. It has received extensive attention. This article designs a neural network FPGA accelerator for XNOR algorithm. The structure of this paper is as follows. The second section introduces the current research and XNOR algorithm; Section third introduces the design and development platform; The fourth chapter introduces the acceleration core; The fifth section analyzes the accelerator performance.

2 Current Neural Network Compression Research and XNOR Network

Recent research of neural network algorithm layer for model compression mainly includes the following methods to improve the deployment performance of neural networks. One is quantized data, and Li et al. [7] uses two bits for data representation or even single-bit data for weighting. This method can reduce the weight of storage, access pressure, and more importantly, it can reduce the hardware consumption required for the calculation. The other is the purpose of reducing the computational cost by reducing the number of weights, removing which values are 0 or weights whose absolute value is close to zero.

2.1 XNOR Net

The XNOR network achieves the purpose of compacting the model by inputting binary values of the weights and feature maps in the convolution calculation process into +1 or -1, which is an extremely quantized network model. Experiments have shown that this method partially reduces the accuracy of the model. The AlexNet provided by the author has 69.2% top-5 accuracy on ImageNet [8]. The XNOR network contains only logic and addition operations in the convolution calculation, which is very suitable for FPGA implementation. Since the algorithm used in the design of the convolution calculation in the neural network adopts the algorithm, the binary convolution part in the AlexNet implementation using the XNOR algorithm is accelerated.

In the XNOR network, a convolutional layer operation can be represented by I *W, where I represents the input, dimension is c*win*hin, W represents the convolution weight, dimension is c*w*h. The XNOR algorithm expects to use a binary convolution kernel B and a scale parameter instead of the original convolution kernel W to obtain the formula:

$$I*W \approx I \oplus a B \quad (1)$$

Where $\oplus$ indicaes a binary convolution, $a \in$ R+, B $\in \{+1, -1\}^n$. Assume

$$J(B, a) = \| W - aB \|^2 = a^2 B^T B - 2aW^T B + W^T W \quad (2)$$

When making equation (2) the optimal solution, since $\{+1, -1\}^n$, then $B^TB=n$, is a constant, and W^TW is also a constant and is a positive number. Then get optimal solution. After obtaining B and deriving B, and letting the derivative result equal to 0, the optimal solution is obtained: $=||W||/n$. Similar processing is performed on the input, and a binary convolution formula is finally obtained as (3), where I is the convolution input tensor, and W is the convolution weight, and the constant matrix obtained by the training.

$$I*W \approx (sign(I) \oplus sign(W)) \odot Ka \quad (3)$$

Table 1 shows the hierarchy of XNOR's AlexNet implementation. Its network has a representative meaning of a neural network because its convolution core size is between 1X1 and 6X6, and the core size of current convolutional neural networks is also generally the case. The XNOR network structure is shown in the following table.

Table 1. Setting Word's margins.

layer	type	layer	type	layer	type
1	Conv	11	Conv	21	Conv
2	BN	12	BN	22	BN
3	ReLU	13	activ	23	activ
4	MP	14	Conv	24	Conv
5	BN	15	BN	25	BN
6	activ	16	activ	26	ReLU
7	Conv	17	Conv	27	Conv
8	MP	18	MP	28	SoftMax
9	BN	19	BN		
10	activ	20	activ		

Conv: Binary Convolution BN: Batch normalization
ReLU: ReLU activ: Binary active

3 Introduction to design platform

This design uses XILINX vivado HLS as the main software for accelerating the core design which can increase hardware design productivity. At present, high-level language development RTL design has become a way for FPGA development. The HLS design method is a high-level language development method that is provided by Xilinx and is based on C-based design synthesis and RTL implementation. HLS is currently recommended as a development method for XILINX， witch can help software engineers use FPGAs to complete complex algorithms.

Figure 1 shows the main flow of HLS design. Implement theme design and test incentives using c or c++ language.

Equations should be centred and should be numbered with the number on the right-hand side.

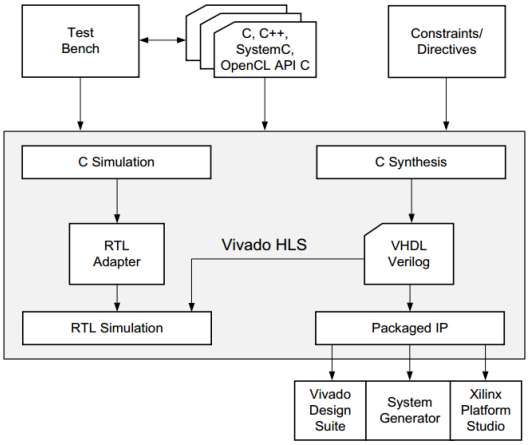

Fig. 1. Vivaso HLS Design Flow

Constraints/Directives are responsible for performing the corresponding RTL code structure optimization. C language design automatically generates RTL code through tool compilation, and the software automatically generates test incentives through TestBench content to complete RTL level behavior simulation. After the main design is completed, you can use the compiler switch provided by HLS to optimize the main design, such as streamlining the design to improve the performance of the solution; use the dataflow option to split the design into small units, each unit independent of each other, improve the design The speed of response. After the test simulation, the design is exported. The software will give the expected hardware resource occupancy, timing analysis, etc., and provide developers with reference.

4 Accelerated IP structure introduction

4.1 Calculation unit division

Our article implements accelerating cores to accelerate the layer 7 to 10 layer of XNOR, which are 5x5 accelerating core. According to the type of operation, the XNOR implementation of AlexNet is divided into full-precision convolutional layer, Binary convolutional layer, Max pooled layer, batch normalization layer, ReLU activation layer, and full-connection layer. The typical calculation includes a normalized layer, an active layer, a convolutional layer, and a pooled layer. The middle layer result will be used in the calculation process and will not be cached. The results of the input and output of the unit are binary results. This will save the storage space. Due to the large number of XNOR networks, this approach can be well reused for unit design and facilitates rapid porting. The accelerator module is divided into different sub-function blocks according to the calculation content. They are data move module, binary convolution module, pooling module, activation and batch normalization module.

* Corresponding author: totemxm@163.com

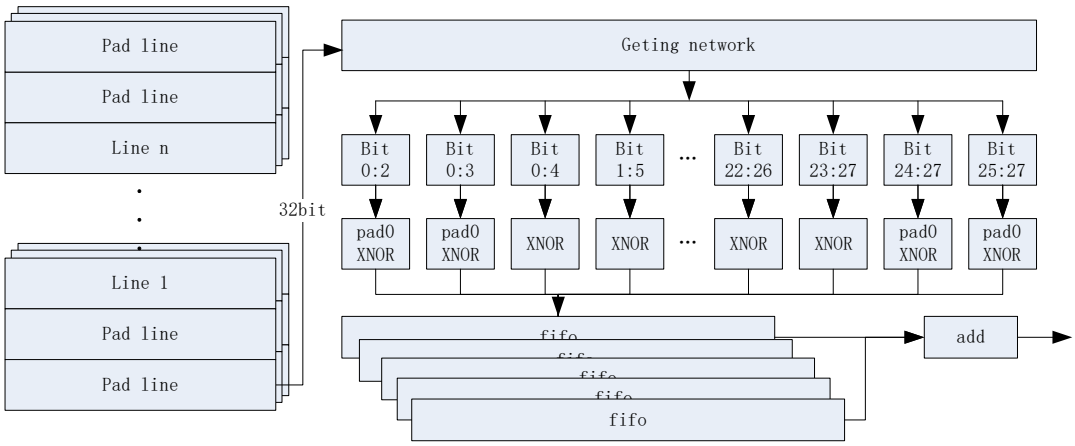

Fig. 2. Binary Convolution Core Block Diagram

4.2 Unit design

A computing core contains multiple sets of weight inputs. Binary convolutions contain weighted rescale factors, biases, and binary weights, where the weights and deviations are of the single-precision floating-point type. Batch normalization includes weights, deviations, running averages, and vector values. There are many parameters, and the ownership parameters are all saved to the FPGA external DDR. They are input into each accelerator core FIFO through unified data load management, and the core moves the required weights from the FIFO. Each feature map will be stored in the ping pong RAM. Each accelerated core has no effect on each other.

The XNOR network takes input and weights into binary and participates in calculations. The most important feature is that the 32-bit data of each floating-point pixel is binarized into 1-bit data, which saves the buffer space required in each stage. A full-precision accelerator design typically inputs one pixel per shot (32-bit floating-point data) and completes the corresponding calculation. If an XNOR network is used for accelerator design, inputting one pixel per shot will make the system no longer efficient. Therefore, this binary convolutional calculation core is designed to read the characteristic map data per cycle and obtain a row convolution calculation result. This design requires only one intermediate sum per c, which reduces hardware complexity and makes the system easier to design. Take an example, one line of feature map data is read per shot. Two zero lines are added before and after the input of the feature map to implement the zero pad operation. When reading the feature map, the data required by each logic operation is selected by the gating network. The edge of the feature map needs to select 3 or 4 bits due to network zero-padding, and the rest selects 5-bit data. The data is logically operated with 5x5 convolution core 25-bit weights of 5 lines, and an 8-bit integer result is obtained based on the number of 1s in the result. Splicing all the integer results to get one line and the result. According to the current line number information, it is finally determined whether to fill in the zero line result or store the current line partial sum result in the fifo. After all the feature maps are calculated, the final convolution result is obtained. In order to improve the performance of the 3x3 accelerated core, the design has increased the parallelism between the convolution windows. The gated data is calculated with multiple convolution kernels.

All AlexNet uses the Max pooling method and its parameters are the same, all using 3x3 pooled cores. Since binarization and floating point calculations are separated in the overall design, only integer comparisons are needed in pooling. The Max pooled input is the complete result after the binary convolution, and the amount of data that needs to be pooled is greatly reduced. The design of the pool is divided into two steps for calculation.

First step, read each row of partial sum results and perform a maximum pooling comparison between each of the three data to obtain row pooling results.

Second step, current number of rows to determine the follow-up operations on the row pooling results. When row 0, only the MAXB is input into MAX1; when an odd row is completed, the MAX1 and the current row pooling result MAXB are completed. Row-pooling operations, and cache the results to the MAX2; when the even-numbered row, the row pooling result MAXB enters the MAX1, and completes the current row pooling results MAXB and MAX2 cache, and output the result.

The batch normalized layer can reduce the sensitivity of the network to the initialization weights, Its mathematical formula is

$$O=(I-BNMean)\times BNWeight+BNBias \quad (4)$$

* Corresponding author: totemxm@163.com

In this design, only the selection action of the pooling layer is performed after convolution, and there is no other data calculation process. Then the calculations in the convolution are pushed to the batch normalization layer. The final calculation in this layer is

$$O = I \times CW \times BNW + (CB - BNM) \times BNW + BNB \quad (4)$$

Where CW is convolution weight, CW is convolution bias, BNW is batch normalization mean, BNW is batch normalization weight, BNB is batch normalization bias. The batch normalization layer reads a row of pooled results, calculates the final floating-point result after the final weights are processed. Compress all single-bit results into next-level input.

5 Performance analysis

The design was implemented on the XILINX Vivado HLS 2016.04 with the K7325T as the target device. Input adopts the picture in ImageNet, after the pretreatment forms the binary characteristic map, and completes according to the row arrangement. Weights use post-training weight data provided by the XNOR network. The design implements XNOR's relevant calculations. Get equal results with the same input. The unit has a convolution kernel size of 5x5. The unit running frequency is 200MHz, and its input feature map is 27*27 and pad is 2. According to the calculation of one line of feature image for each shot, ideally, 31*96*256=761,856 cycle are required to complete its convolution calculation. According to the HLS simulation result, 5x5 convolution is completed using 813,568 cycle, and the total running time is 4.06ms. 93.6% for the ideal situation. The other core's resources is list in Table 2. In Table 2, we use the clock and throughput to evaluate the performance of the design.

Table 2. Comparison Resources and runtime

core	Platform	Clock(MHz)	Throughput(FPS)	BRAM	DSP	kLUT	GOPS
[6]	XC7Z020	200	106	105	89	44	410.22
[5]	XC7Z020	143	420	32	1	14.5	329.47
Our	K7-325T	200	246.3	133	66	20.5	302.2

6 Conclusion

This article uses the XILINX Vivado HLS tool to implement an FPGA accelerated design of the binary convolutional part of the AlexNet implementation of the XNOR algorithm. The Accelerated Core implements the relevant calculations of the XNOR algorithm. Simulation with XILINX K7-325T as the target device. Get the design resource consumption and execution time. Compared to other traditional computing platforms, our FPGA platforms design provide more flexible deployment capabilities.

REFERENCES

1. Xingchen Zhou. Deep Model Based Offline Handwritten Chinese Character Recognition[D]. ZheJiang University,2016.
2. Zhiqiang Liu, Yong Dou, Jingfei Jiang, Jinwei Xu, Shijie Li, Yongmei Zhou, and Yingnan Xu. 2017. Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks. ACM Trans. Reconfigurable Technol. Syst. 10, 3, Article 17 (July 2017), 23 pages. DOI: 10.1145/3079758
3. Mingxing Duan, Kenli Li, Canqun Yang, Keqin Li, A hybrid deep learning CNN–ELM for age and gender classification, Neurocomputing, Volume 275,2018,Pages 448-461,ISSN 0925-2312,DOI:10.1016/j.neucom.2017.08.062
4. Yu Zijian. FPGA-based Accelerator For Convolutional Neural Network[D].ZheJiang University,2016.
5. Hiroki Nakahara, Tomoya Fujii, and Shimpei Sato. 2017. A fully connected layer elimination for a binarizec convolutional neural network on an FPGA. In Field Programmable Logic and Applications (FPL), 2017 27th International Conference on.IEEE, 1–4
6. Li Jiao, Cheng Luo, Wei Cao, Xuegong Zhou, and Lingli Wang. 2017. Accelerating low bit-width convolutional neural networks with embedded FPGA. In Field Programmable Logic and Applications (FPL), 2017 27th International Conference on. IEEE, 1–4.
7. Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary weight networks. arXiv preprint arXiv:1605.04711 (2016).
8. M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNORNet: ImageNet Classification Using Binary Convolutional Neural Networks. European Conference on Computer Vision (ECCV), Oct 2016. arXiv:1603.05279.