Research on Parallel Acceleration of Chebyshev Approximation based on Neural Network

Wang Xiangchun[1], Chen Zijian[2], Wang Bingchao[3]

[1]School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, 200240

[2]School of Chemical Engineering, Dalian University of Technology, Panjin, Liaoning, 124221

[3]Tongda College of Nanjing University of Posts & Telecommunications, Yangzhou, Jiangsu, 225000

Keywords: Deep neural network; Image reconstruction; parallel computing; Embedded GPU; Chebyshev approaching

Abstract: It is difficult to deploy and implement limited neural network computing (Deep Neural Network, DNN) on embedded GPU with limited resources and solving ability. The neural network needs parallel accelerated design because of its huge amount of data and high computational complexity of the convolution layer. In this paper, Chebyshev polynomial is used to approximate the convolution kernel, and the optimization scheme is applied to the deep neural network for image reconstruction to realize the parallel processing of convolution operation and reduce the computational complexity. Then the parallel accelerated design based on GPU is carried out for the optimized network convolution layer. Finally, the whole network is transplanted to the NVIDIA AGX Xavier embedded development board to realize the reasoning process of image reconstruction. The experimental results show that the reasoning speed of the parallel accelerated network reconstruction is 2.2 times faster than that of the original network.

1. Introduction

In recent years, Deep Neural Network (DNN) has achieved great success in image classification, recognition, detection and other fields. However, DNN is generally characterized by large data volume and high computational complexity, which is particularly prominent in the convolutional layer [1], which makes it difficult for the network to be transplanted to the embedded development board with limited resources. However, with the penetration of high-tech equipment into every aspect of life, it is becoming more and more common and important to realize computation-intensive applications on many devices.

Therefore, the reasoning of DNN on resource-limited platforms has become the focus of researchers. In order to rapidly deploy DNN in embedded terminals, researchers have proposed various schemes. In 2015, Han Song et al. [2] proposed to remove redundant network connections and adjust the weight to achieve network compression, so that the network can be deployed on embedded mobile application terminals. Literature [3] compressed and highly optimized the network model, transformed the network from a heavyweight model to a lightweight model that can be deployed on the embedded development board, and realized a real-time driver sleepiness detection technology based on GPU embedded system. Literature [4] proposes a real-time stereo collision detection adaptive system based on embedded GPU by adopting sensor fusion ultrasonic to better filter noise and combining GPU hardware characteristics. In literature [5], asynchronous multithreading parallel computing is adopted to implement smile detector on GPU embedded platform. A modular scheme is adopted to prioritize threads and load balance among tasks is achieved by increasing the number of threads of critical tasks. Mohammad team for mobile terminal within DNN reasoning were studied many [6], found that according to the equipment resource reasonable using different thread granularity parallel design can significantly increase the speed of network reasoning, and puts forward the data rearrange, not the precise calculation and data

  DOI: 10.25236/isaiee.2020.011

processing scheme to realize the network speed. DNN for detection, identification and other tasks is gradually applied to mobile devices with limited resources.

Chebyshev polynomial can achieve the best approximation of function, so this article USES the Chebyshev polynomial convolution kernels for parallel processing of network layer to simplify convolution calculation, the optimization scheme used for reconstructing the depth of the neural network, then after optimization of the network convolution layer for concurrent design based on GPU. In order to ensure the universality of the scheme, the image reconstruction network based on manifold learning proposed by Zhu, which can be applied to a variety of different image reconstruction tasks. Finally, the network after parallel design was transplanted to NVIDIA AGX Xavier embedded development board, and its performance was analyzed.

2. Manifold learning depth neural network for image reconstruction

Based on the depth of the manifold learning neural network structure as shown in Figure 1, the sensor data as input of the link layer 1, due to the sensor data is a plural, commonly exist real part and imaginary part of complex sensor data for $n \times n$, the need to convert them into $2n^2$ real numerical input connection layer 1, so the whole link layer 1 also said network input layer, neuron number $2n^2$.The number of neurons in all connection-layer 2 and 3 was n^2, and tanh activation function was used. In order to match the convolution operation, the output data of the full connection layer 3 is converted into dimension, from $n^2 \times 1$ to $n \times n$.Convolutional layer 1 uses 64 5×5 convolution kernels for convolution operation at step 1, while convolutional layer 2 uses 64 5×5×64 convolution kernels for convolution operation at step 1.*Relu* was adopted as the activation function in all the convolutional layers. Finally, a 7×7×64 convolution kernel was adopted in the convolutional layer 2 for deconvolution and the reconstructed image was obtained with the step size of 1.

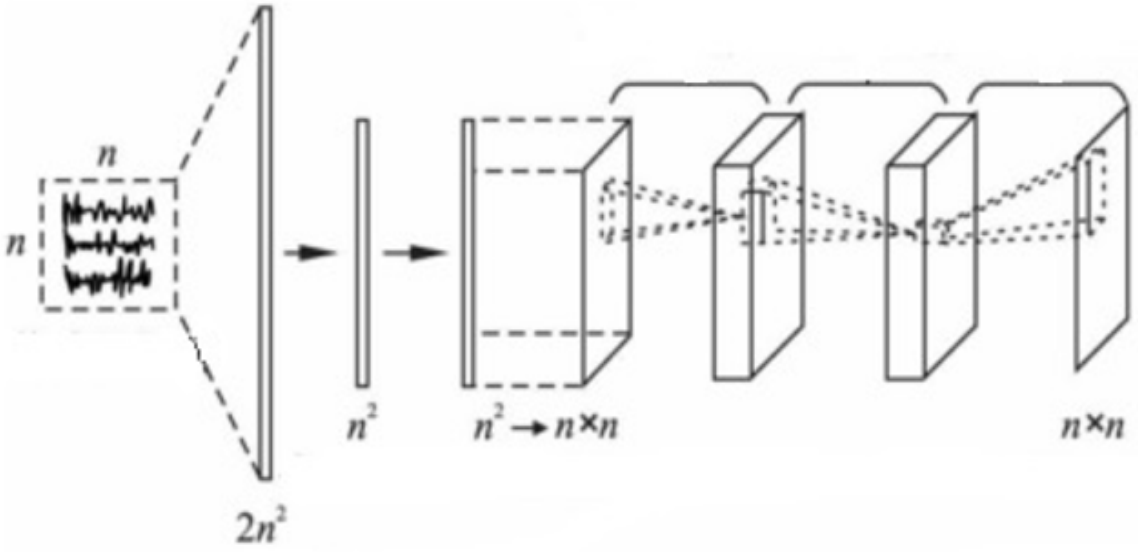

Fig 1. Structure of the network

3. Convolution layer optimization based on Chebyshev polynomials

3.1 Chebyshev approximation principle

The core of approximate computation lies in the ability to give the approximate optimal solution of the problem in polynomial iterative time. Polynomials can realize function approximation and transform the problem into the study of polynomial class. And because the sum, difference and product of polynomials are still polynomials, the complex computation can be realized quickly through iterative rules.

Chebyshev polynomials can achieve the best uniform approximation. Considering that it is often necessary to solve a known complex function *f(x)* in practical applications, in order to simplify the calculation, it is usually necessary to find a function $Q_n(x)$, so that the error between the two can be minimized in a certain metric sense. In Chebyshev's best consistent approximation theory, the function $Q_n(x)$ is Chebyshev polynomial, and the difference between a certain interval [a, b] and *f(x)* is the smallest interpolation between all polynomials in the interval *Q(x)* and *f(x)*, as shown in Formula (1):

$$\max_{a\le x\le b} | Q_n(x) - f(x) | = \min | \max | Q_{a\le x\le b} - f(x) \quad (1)$$

The function approximation theory of Chebyshev polynomials [13] shows that such a polynomial $Q_n(x)$ exists and is unique. Set $E_x = \max_{a\leqslant x\leqslant b} | Q_n(x) - f(x)|$, when *E(x)* exists at least n+2 interleaved points [$x_1 \ldots x_{n+2}$] on [*a, b*] ($a \le x1 < \ldots < xn+2 \le b$), so that *E (xi)* = ±*En*, where I ∈ [*1, n+2*], $Q_n(x)$ is the best uniform approximation of *f(x)*.

3.2 Chebyshev polynomial approximation of convolution kernel

Because Chebyshev polynomials can achieve the best uniform approximation, Chebyshev polynomials are considered to optimize the convolution kernel. Assume that the input image size is 3×3 and the convolution kernel size is 2×2(for the sake of simple demonstration here, the convolution kernel size is actually less 2×2).

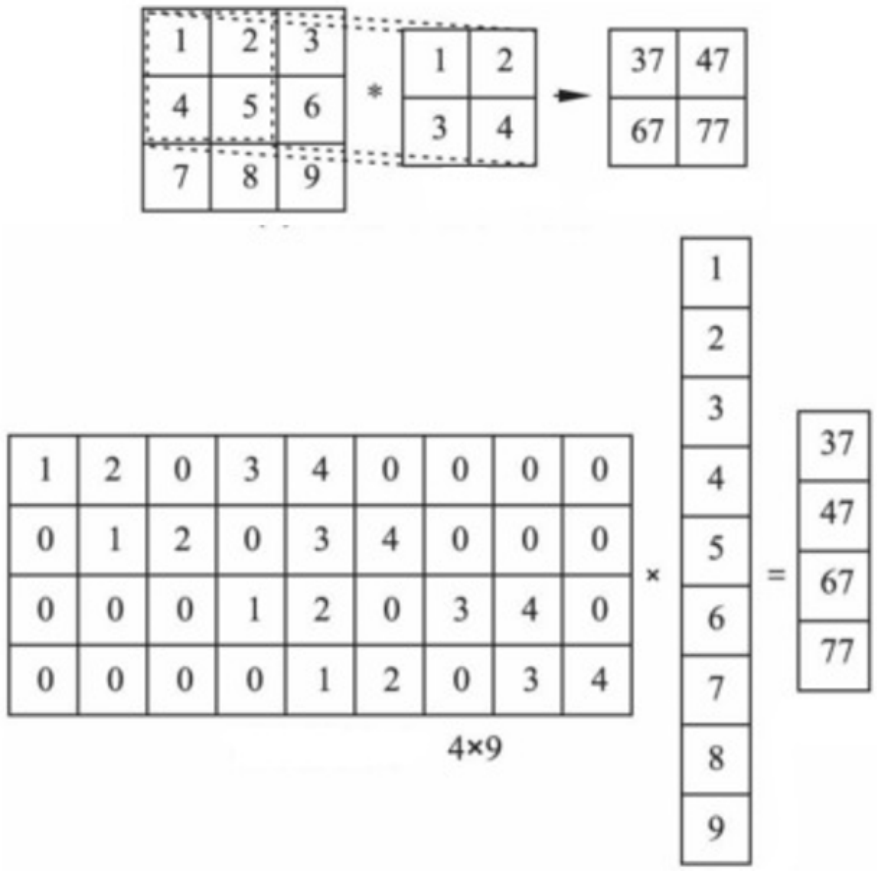

Fig 2. Convolutional operations

In the original convolution operation, the convolution kernel size is generally small, and only part of the features can be extracted each time. It is often necessary to traverse each input feature map, and finally carry out information integration in higher dimensions, with a complex calculation process.

3.3 Parallel acceleration design of convolutional layer

CUDA architecture is GPU heterogeneous programming architecture launched by Nvidia. In this paper, CUDA architecture is combined to carry out parallel design of network convolutional layer. The key lies in the parallel design of Chebyshev polynomial iteration.

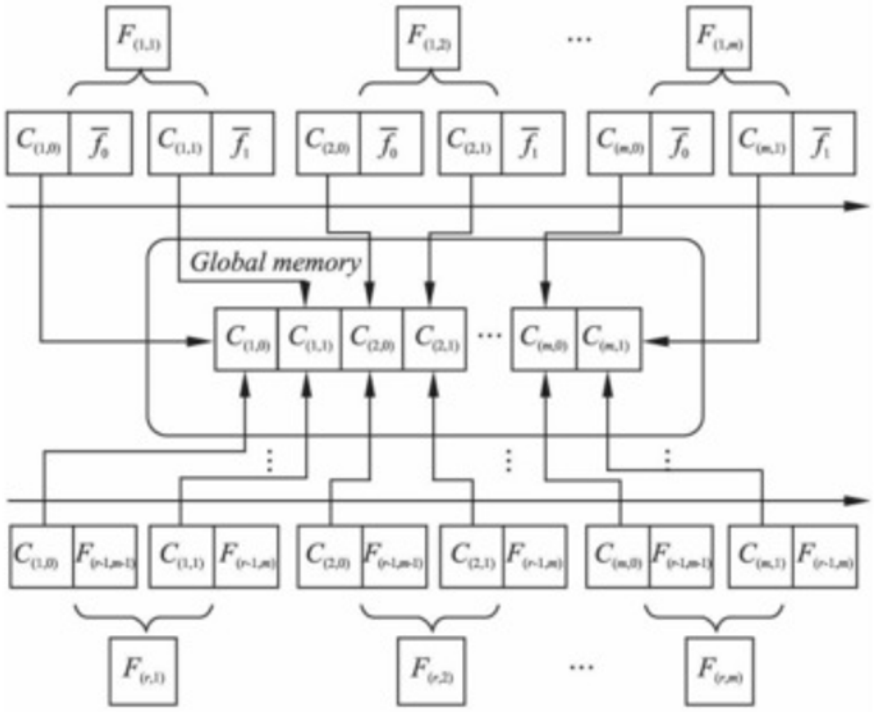

Fig 3. Process of Chebyshev iterative

In CUDA acceleration design, thread parallelism is m. In subsequent iterations, the corresponding polynomial coefficient calculated in advance is directly called from GPU global memory to calculate the value of the corresponding polynomial, so as to avoid the repeated calculation of the polynomial coefficient value in each iteration and reduce the number of data transmission between GPU and CPU ends. Figure 4 shows the process of CUDA parallel processing. Two pieces of memory are opened out in the global memory, one for storing the calculated polynomial coefficient, and the other for storing the calculated polynomial value. At the beginning of iterative computation, the thread block reads polynomial coefficients from global memory and allocates corresponding values for the threads within the block. At the end of each iteration, the thread broadcasting mechanism is adopted to broadcast the polynomial values calculated by the last two threads within the block as the basis vector for the next iteration.

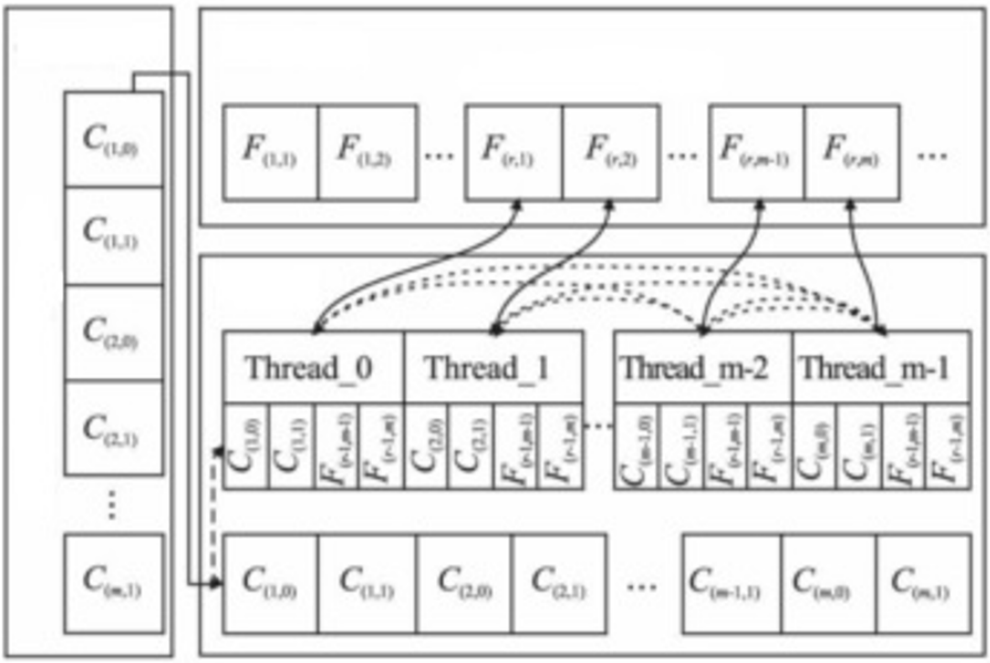

Fig 4. Cuba-based Chebyshev iterative parallel design

Using the above parallel scheme, *k* values only need to be iterated *k/m* times, which greatly reduces the computation time. In addition to the first round of iteration needs complex polynomial coefficient calculation, the subsequent iterations of each *fk1* calculations are only needs two multiplication subtraction.

The parallelism m can be flexibly set according to the hardware platform resources. Since all polynomials in each round of iteration are calculated at the same time, the calculation time in each round of iteration is obtained by two multiplications and one subtraction operation, so the time required for each iteration is fixed, denoted as *Time1*.Assuming that the parallelism is increased to m+ E, the required number of iterations becomes *k/(m+E)*, and the reduced time in the iteration process after increasing the parallelism is *(k/m-k/(m+e))×Time1*.At the same time, the number of polynomial coefficients to be calculated before the iteration increases from 2m to *2(m+e)*. In order to ensure that the increase of parallelism can bring the gain of computing time, two points need to be paid attention to at this time:

1) The polynomial coefficient needed to be calculated increases, and whether the hardware platform has enough thread resources to support their calculation, without waiting for the previous calculation to finish and completing the resource release;

2) The calculation time of the increased polynomial coefficient is denoted as Time2, and *Time2≤(k/m-k/(m+e))×Time1* should be guaranteed. By setting the parallelism according to the above two considerations, we can make full use of the resources on the hardware. In this paper, the thread parallelism degree m of the Chebyshev polynomial iteration process is set to 3. Compared with the serial iterative process, parallel computing is more flexible, and the computing efficiency can be significantly improved.

4. Experimental analysis

The model was trained by using the medical brain image data set from the *mgh-uschcp* public data set library. The image size of the data set was processed into 64 × 64. The same training environment was used before and after optimization: The CPU was Intel(R) Xeon (R) E5-4610 V4, and the total memory was 15G.GPU is Tesla P100 with 256G video memory. The Tensor flow version is 1.8; CUDA version is 9.0.The inference environment used for network reconstruction verification after optimization: CPU is Core I7-7700 and memory is 32G; GPU is Nvidia GeForce GT 730, and its video memory is 2G.CUDA version is 9.0.The optimized model was compared with

the original model to verify the reconstruction performance of the optimized model, and the Peak Signal to Noise Ratio (PSNR) was used to evaluate the reconstructed image quality. Then the optimized model was mounted on the Jetson AGX Xavier embedded platform and the acceleration effect was analyzed.

4.1 Verification of network model reconstruction after optimization

The model training methods before and after optimization are consistent, the batch data size is set as 100, the learning rate is 0.00002, and the RMS Prop algorithm is used to realize gradient descent to accelerate the convergence speed. The mean square error is used as the loss function. FIG. 5 shows the loss-iteration curves of the two models in the training process. It can be seen that the initial loss value of the optimized model is larger than that of the original model. As the number of iterations increases, the two curves get closer and finally achieve similar reconstruction performance.

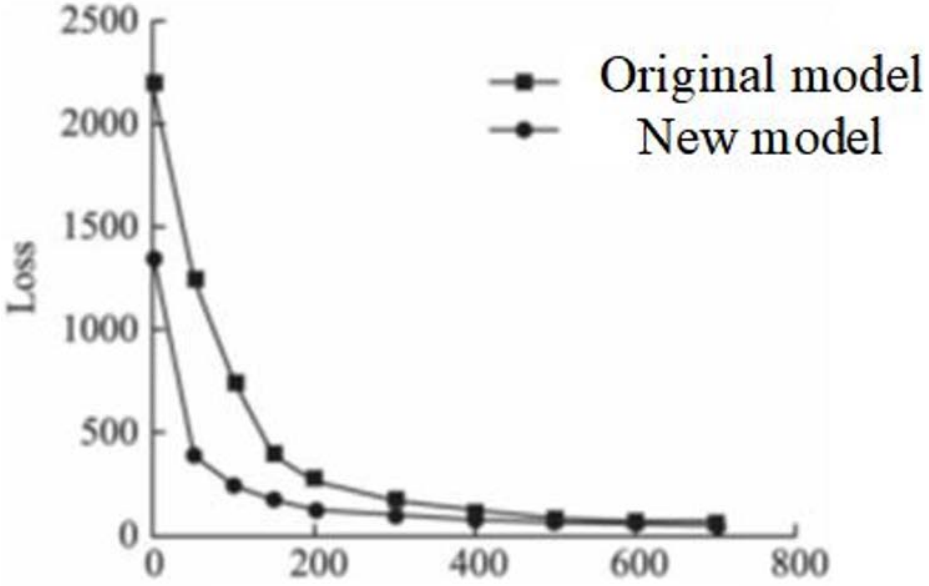

Fig 5. Loss-iteration curves of the two models

4.2 Analysis of computational complexity before and after network optimization

The time complexity and space complexity of DNN are important indexes to evaluate network model architecture. The time complexity determines the time of training and reasoning. When the time complexity is high, the model training time is long, so that the researchers' ideas cannot be verified quickly, and the improvement time of the model will be prolonged accordingly. Therefore, the time complexity of network is an evaluation index concerned by researchers, and is also an important index in approximate calculation. The space complexity determines the number of parameters of the network model. It is necessary to consider whether the limited resources of the hardware platform can meet the requirements of network deployment when carrying out hardware migration. Since the time complexity in DNN mainly comes from the network convolutional layer, the time complexity analysis of the convolutional layer is mainly carried out. Formula (11) represents the time complexity of a single convolutional layer, where M represents the size of the output feature graph, S represents the size of the convolution kernel, and C_{in} and C_{out} respectively represent the number of input and output channels of the convolutional layer:

$$Comlpex_{Time} = O(M^2 \times S^2 \times C_{in} \times C_{out}) \quad (2)$$

$$Comlpex_{Time} = O((S^2 \times C_{out})^2) \quad (3)$$

According to formula (2) and formula (3), the time complexity of each layer of the network before and after optimization can be calculated, as shown in Table 1.Convolutional layer 1 and deconvolutional layer are the input and output layers of the convolution part. Due to the situation that the number of channels in the input or output layer is 1 in the network of these two layers, the time

complexity of the optimized model convolutional layer 1 and deconvolutional layer 1 does not decrease much, but the time complexity of convolutional layer 2 decreases 125.44 times. It can be concluded that significant time complexity reduction can be achieved through the complexity of the convolutional layer hidden in the middle of the network after optimization.

5. Conclusion

To sum up, this paper proposes to use Chebyshev polynomials to approximate the network convolution kernel, aiming at the problems of high data load and high computational complexity of network convolution layer in deep neural networks. By expanding the convolution kernel, the convolution operation with the input data is converted into the form of matrix vector multiplication, and the convolution result can be calculated directly without traversal operation. Then, the Chebyshev polynomial is used to perform iterative fitting for the parameters in the convolution kernel, so that the convolution operation can realize parallel processing. On the basis of network optimization, a corresponding parallel acceleration scheme is designed for Chebyshev polynomial iteration of convolutional layer based on CUDA programming architecture. Finally, the network was ported to the NVIDIA Jetson AGX Xavier embedded development board, where the reasoning time was only 0.35s. Compared to the original network reasoning can achieve 2.2 times the acceleration.

References

[1] Chellapilla K, Puri S, Simard P. High performance convolutional neural networks for document processing[C]// 10th International Workshop on Frontiers in Handwriting Recognition,2006.

[2] Han S, Pool J, Tran J, et al. Learning both weights and connections for efficient neural network[C]//Advances in Neural Information Processing Systems, 2015:1135- 1143.

[3] Reddy B, Kim Y H, Yun S, et al. Real- time driver drowsiness detection for embedded system using model compression of deep neural networks[C]// Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017:121- 128.

[4] Stanoev A, Audinet N, Tancock S,et al. Real- time stereo vision for collision detection on autonomous UAVs[C]// 2017 IEEE International Conference on Imaging Systems and Techniques (IST),IEEE,2017:121-128.

[5] Ghazi P, Happonen A P, Boutellier J, et al. Embedded implementation of a deep learning smile detector[C]// 2018 7th European Workshop on Visual Information Processing (EUVIP), IEEE, 2018:1-6.

[6] Motamedi M, Fong D, Ghiasi S. Fast and energy- efficient cnn inference on iot devices[J]. arXiv preprint arXiv:1611. 07151, 2016.