FPGA-based Acceleration of Neural Network Training

Ruoyu Sang, Qiang Liu, and Qijun Zhang

School of Electronic Information Engineering, Tianjin University, Tianjin, Nankai, 300072, China

***Abstract* — Neural networks (NNs) have been widely used in microwave device modeling. One of the greatest challenges is how to speed up the model training process and reduce the development cost. To address the issue, this paper exploits FPGAs to accelerate NN training. Experimental results demonstrate that the model training time can be reduced by up to 99.1%, compared to the traditional software implementation.**

***Index Terms* — neural network, quasi-Newton method, FPGA, hardware acceleration.**

I. Introduction

Nowadays, neural network (NN) techniques have a broad range of applications in microwave modeling problems, such as coplanar waveguide components, transistors, amplifiers, and bends [1]. Training is a critical subtask in developing a neural network model and quasi-Newton (QN) method is used for this purpose in particular [2].

Conventionally, NN training is implemented on software. However, due to the complex matrix and vector operations in QN, iterative nature of optimization and worse parallelism of software, the training consumes a large amount of computation time. Therefore, it is essential to consider a customized hardware implementation to improve the training speed, so that optimized microwave models can be obtained in limited time. Recently, with the increasing density and large amounts of embedded arithmetic blocks, modern high-capacity Field Programmable Gate Arrays (FPGAs) have been considered to be a more attractive alternative to accelerate scientific applications. There are a number of research works on accelerating optimization algorithms using FPGAs, QR-decomposition [3] and Conjugate Gradient [4]. By customizing architecture around algorithms, FPGA-based hardware implementations can achieve speed much faster than software implementations. This enables neural network training to be implemented in reconfigurable and custom hardware to reduce the cost of neural model development.

This paper reports recent FPGA-accelerated QN implementations which significantly accelerate NN training and the microwave modeling process.

II. Preliminary

A. NN Training and QN

The objective of NN training is to minimize the error function with respect to network weights $\boldsymbol{w}$, which can be defined as

$$\min E_T(\mathbf{w}) = [\frac{1}{|S_T|N_3}\sum_{m=1}^{|S_T|}\left|\frac{\mathbf{o}_m - \mathbf{d}_m}{d_{max} - d_{min}}\right|^2]^{1/2} \qquad (1)$$

where $\boldsymbol{o}$, $\boldsymbol{d}$ and $\boldsymbol{w}$ represent output vectors, desired output vectors and weights vectors of the NN structure, respectively. $|S_T|$ is the number of training sample $\{m \in |S_T|\}$. d_{max} and d_{min} are the maximum and minimum values in desired outputs.

The most efficient algorithm for solving the problem in (1) is the QN method, which is illustrated in Algorithm 1.

Algorithm 1 QN method

1. $k = 0$, $\varepsilon = 10^{-3}$;
2. Initialize $\boldsymbol{w}_0 \in \boldsymbol{R}^n$, $\boldsymbol{H}_0 = \boldsymbol{I}$, $\boldsymbol{g}_0 = \partial E_T(\boldsymbol{w}_0)/\partial \boldsymbol{w}$;
3. **While**($k<k_{max}$ or $\|\boldsymbol{g}_{k+1}\| \leqq \varepsilon$)
4. Calculate search direction $\boldsymbol{d}_k = -\boldsymbol{H}_k\boldsymbol{g}_k$;
5. Decide a step size λ_k by golden section method;
6. Set $\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \lambda_k\boldsymbol{d}_k$;
7. Calculate gradient $\boldsymbol{g}_{k+1} = \partial E_T(\boldsymbol{w}_k)/\partial \boldsymbol{w}$;
8. Set $\boldsymbol{s}_k=\boldsymbol{w}_{k+1}-\boldsymbol{w}_k$, $\boldsymbol{y}_k=\boldsymbol{g}_{k+1}-\boldsymbol{g}_k$, update $\boldsymbol{H}_{k+1}$ in QN formula;
9. $k = k + 1$;
10. **End**

B. FPGA Basis

FPGA is an integrated circuit designed to be configured by designers after manufacturing. The architecture consists of an array of programmable logic blocks, I/O pads, and routing resources. Moreover, contemporary FPGAs contains dedicated blocks fixed into the silicon. Examples of these include DSP blocks, Block RAMs (BRAM), multi-gigabit transceivers and processor cores. These resources provide FPGA with the software-like design flexibility and, more importantly, the capability of customizing computing systems around the algorithms.

III. FPGA Implementation of QN

The FPGA-based QN architecture is sketched in Fig.1, which is comprised of six modules: computing schedule controller (CSC), pseudorandom number generator (PNG), line search (LS), gradient computation (GC), $\boldsymbol{H}$ updating (HU) and neural network evaluation (NE).

CSC, with a finite state machine, is utilized to schedule the operation sequence of all modules and communicate between memories and corresponding modules. PNG generates random initial values $\boldsymbol{w}_0$ based on 32-bit linear feedback shift registers. LS obtains step size λ_k by the golden section search algorithm. GC computes the gradient $\boldsymbol{g}_k$ and executes termination test.

978-1-4673-8762-0/16/$31.00 ©2016 IEEE

HU updates $\boldsymbol{H}$ in QN formula and determines a search direction. NE implements the neural network model.

Two designs of GC are explored in this work. The first design (Design 1) is the general approximate calculation method, which is applicable various object functions, but is time-consuming due to frequent evaluations of NN. Another (Design 2) is the gradient expression particularly deduced from the NN training, which consists of a series of add, subtract and multiply operations and facilitates FPGA implementation.

In addition, all intermediate computation results are buffered in FPGA on-chip dual-port RAMs for fast accesses, while all training data are stored in off-chip DDR3 SDRAMs.

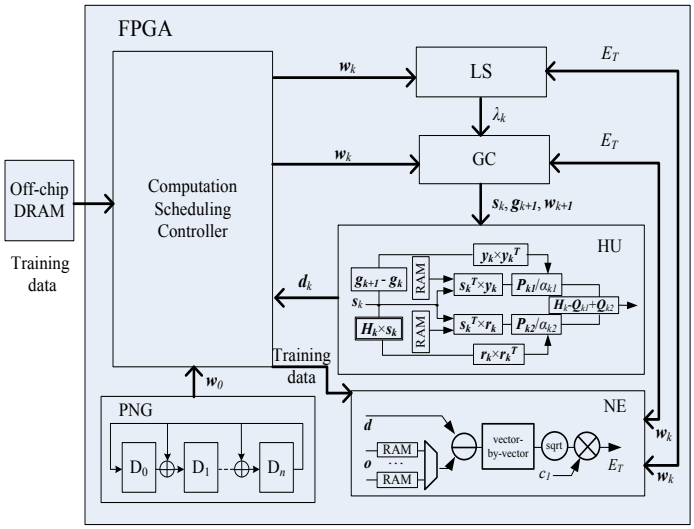

Fig. 1. Block diagram of FPGA-based QN architecture.

IV. Experiment Result

The proposed QN designs are synthesized and implemented on xc7vx690t FPGA. The maximum frequency of 250 MHz is achieved.

Table 1 gives the resource usage of the four key modules of two hardware implementations for 3-8-1 NN, in terms of flip-flop (FF), look-up-table (LUT), DSP and Block Ram (BRAM). Vertically, NE requires largest amount of resources due to complicated calculation of training error function, and HU comes second owing to plenty of matrix-vector operations. Horizontally, since more complex arithmetic operations, the second GC design consumes more resources than the first one.

Table 2 compares the performance between two hardware implementations and equivalent software implementations. The software versions are executed on Intel Core i5-4590 CPU in C++ code. Five NNs with different structures are trained with 512 training data. This paper takes the average execution time of single iteration of QN as performance metric. From the table, we can make the following observations. First, the speedup of 17 times is achieved by Design 1 and 106 times is obtained by Design 2, compared to the corresponding software implementations. Second, Design 2 shows scalable performance as NN size increases, i.e. the execution time increase is much slower than the size increase.

TABLE 1
Resource Utilization of Each Module

Neural Network 3-8-1	Design 1				Design 2			
	FF (%)	LUT (%)	DSP (%)	BRAM (%)	FF (%)	LUT (%)	DSP (%)	BRAM (%)
LS	0.35	0.60	0.82	0.64	0.27	0.53	0.28	0.51
GC	0.25	0.47	1.09	0.53	0.59	1.23	0.68	0.83
HU	1.21	2.70	0.54	0.20	0.98	1.97	0.68	0.16
NE	2.90	5.14	0.14	4.58	3.51	7.11	3.40	3.05

TABLE 2
Performance Comparison For Various NNs

Neural Network	Design 1 (ms)	Software 1(ms)	Speed-up 1 (×)	Design 2 (ms)	Software 2 (ms)	Speed-up 2 (×)
3-8-1	2.15	36.82	17.12	1.19	32.93	27.67
7-8-1	4.07	70.74	17.38	1.52	63.58	41.83
6-8-10	8.84	155.40	17.58	2.58	139.99	54.26
4-16-12	22.76	401.26	17.63	4.35	361.70	83.15
15-32-1	52.47	928.19	17.69	7.95	844.05	106.17

V. Conclusion

This paper has presented hardware-accelerated neural network training. Compared to software implementations, hardware designs represent a significant speed-up, which can facilitate the model development in the microwave fields.

Acknowledgement

The authors wish to acknowledge the support of National Natural Science Foundation of China (61574099) and National Science and Technology Major Project of China (2012ZX03004008).

References

[1] V. Devabhaktuni, M. C. E. Yagoub, and Q. J. Zhang. "A robust algorithm for automatic development of neural network models for microwave applications," *2001 IEEE MTT-S International Microwave Symposium Digest,* Phoenix: IEEE, 2001:2087-2090.

[2] C. A. Popa, "Quasi-Newton learning method for complex-valued neural networks", *2015 International Joint Conference on Neural Networks (IJCNN)*. Killarney: IEEE, 2015: 1-8.

[3] M. Abels, T. Wiegand, S. Paul, "Efficient FPGA Implementation of a High Throughput Systolic Array QR-Decomposition Algorithm", *Signals, Systems and Computers(ASILOMAR)*, Pacific Grove: IEEE, 2011: 904-908.

[4] A. Roldao, G.A. Constantinides. A high throughput fpga-based floating point conjugate gradient implementation for dense matrices. *ACM Transactions on Reconfigurable Technology and Systems,* 2010, 3(1): 1