Implementation of Finite Element Neural Network Computing Method Based on FPGA

Zhendong Liang[1]; and **Haibin Li[1]**

[1]School of Sciences, Inner Mongolia University of Technology, Hohhot, China

Sp. Issue 2013/ # Paper 9
Corresponding author:
Haibin Li
Mailing address: No. 49, Aimin Street, Hohhot City, Inner Mongolia, 010051, P. R. China
E-mail: lhbnm2002@163.com
Tel: +86-13674786672
Fax: +86-471-6575713

KEYWORDS

Finite element; Neural network computing; Floating -point operations; FPGA; Hardware description language

ABSTRACT

Based on the research of neural network (NN) computing method of finite element equations for the structure of the discrete model, a complete implementation of the various parts of the computing method is presented. The computation module was described with hardware description language (Verilog HDL) at the RTL level, and functional simulation was executed by Quartus II 11.0 and ModelSim 6.5. Simulation results show that the implementation of neural network computing method based on field programmable gate array (FPGA) can satisfy the requirement of the parallel computing, which could further improve the speed of structural finite element computation, and ensure the speed and accuracy of the computation.

Introduction

Finite element method has been developed rapidly during the last century and has become one of the most effective structural analysis tools today. However, with the analysis of structural problems becoming more and more complex and large-scale, finite element computation method based on the traditional serial computer with limited memory capacity and speed becomes inefficient (Liu, *et al.*, 2005; Zhang, *et al.*, 2010; Wang, *et al.*, 2010; Liu, *et al.*, 2010). To overcome the drawbacks, many structural analysts and mathematicians have committed themselves to develop more efficient finite element analysis methods. Hence the theory based on the parallelism of many practical application procedures was proposed.

Neural network is a complex nonlinear system with highly parallel computational capability, and its application domain has been expanded to structural finite element analysis. In the past, the analog circuits were often used in implementation hardware, but it possesses low accuracy and complex structure design. Based on structural finite element analysis of discrete models, a neuro-computing strategy is introduced in this paper, while dynamic iterative equations are constructed in terms of neural networks of discrete models. Determination of the iterative step size, which is important for convergence, is investigated based on the positive definiteness of the finite element stiffness matrix. Consequently, a method of choosing the step size of dynamic equations is proposed and the computational formula of the best step size is derived (Li, *et al.*, 2004; Gao, *et al.*, 2008). The analysis of the computing model shows that the solution of finite element system equations can be obtained with the method of neural network computation efficiently. Thus the proposed method can be used for parallel computation of structural finite element in a large-scale integrated circuit.

In order to ensure the speed and accuracy of the finite element parallel computing, we use the hardware of field-programmable gate array (FPGA) for configuration and simulation of the neural network of the finite element in this paper.

Neurocomputing Method of Structural Finite Element Analysis of Discrete Models

Digital circuits process numerical information. The characteristic of the digital signal in the time scale or size scale is discrete. Therefore, the corresponding

neural network system is also a discrete system. Because the finite elements are assembled by the unit combination, the total potential energy and the restraint of the system can be written in the following from (Consolazio, 2000):

$$\min_{x\in\Omega} \prod = \frac{1}{2}\delta^{\mathrm{T}}\mathbf{K}\delta - \mathbf{q}^{\mathrm{T}}\delta$$
$$\text{s.t} \quad \mathrm{A}\delta = \overline{\delta} \qquad (1)$$

where K is the global stiffness matrix; δ is the solution vector; q is the nodal loading array; A is the constraint matrix; $\overline{\delta}$ is the displacement constraint array and S_u is the constrained surface.

If the constraint is not considered , Eq. (1) can be transformed into the following unconstrained optimization problem (Li, *et al.*, 2004):

$$\min_{x\in\Omega} \prod = \frac{1}{2}\delta^{\mathrm{T}}\mathbf{K}'\delta - \mathbf{q}'^{\mathrm{T}}\delta \quad (2)$$

where $\mathbf{K}'$ is the global stiffness matrix, and $\mathbf{q}'$ is the loading matrix. As the finite element theory, $\mathbf{K}'$ is a symmetric positive definite matrix, and the superscript T denotes the transpose operator.

Taking the right-hand side of Eq. (2) as the neural network energy function, the finite element solution could be obtained by constructing the following discrete dynamic equation of neural networks.

$$\delta^{k+1} = \delta^{k} + (-\mathbf{K}'\delta^{k} + \mathbf{q}')\Delta t \qquad (3)$$

where k indicates the k th iteration and Δt is the step size.

Implementation of FPGA

The discrete NN computational model shows that NN computing structure is composed of many adders and multipliers. Since the computation needs a large number of floating-point arithmetic calculations, the implementation of the floating-point adders and multipliers are flowing in the first. According to the floating-point number format standard IEEE-754, the design of the single-precision 32-bit format (Li, *et al.*, 2010) is achieved, shown in Table 1.

Table 1: IEEE-754 Floating-Point Format

Floating-point	Sign	Exponent	Mantissa
Single-precision 32-bit	[31] 1 bit	[30:23] 8 bit	[22:0] 23 bit

According to the IEEE-754 standard, 0 indicates a positive sign bit, and 1 indicates a negative value. The offsets of 127 units are contained in the order code offset, which represents the range from -127 to 128. The integer bit format is hidden in the mantissa among which 23 units are decimal fraction.

(1) Floating-point Adder Module Implementation

A module of the floating-point adder is shown in Fig. 1, where CLK is a clock signal, Start_Sig module activation signals for the low-level reset signal. A, B, and RSTn input 32-bit single-precision floating-point numbers, and the result is the output of the 32-bit calculation result and Done_Sig is the completion of the feedback signal (IEEE Std 754-2008).

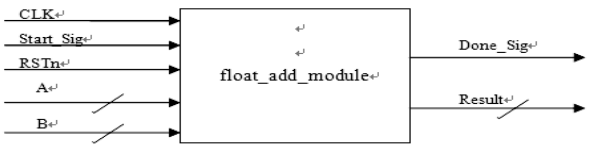

Figure 1: Floating-point Adder Module

The following steps are required to achieve floating point addition.

(1) The preprocessing of input operand. The standard format of the operand sign bits, the exponent bits and the mantissa bit have to be separated. In order to ensure the accuracy of the calculation result, we fill the 23 bits "0" behind the mantissa, two "01" ahead of mantissa, and two "00" before the exponent.

(2) Exponent operation. Obtaining a difference between the two exponents, determining the size and selecting the large one and shifting mantissa alignment according to the difference.

(3) The mantissa Operation processing. According to the sign bit of the operand, and then proceed to the adder.

(4) Calculation results processing. The size of the symbol bits of the operand is determined by exponent. To ensure that the first bit of the mantissa is 1, then to shift operation, and to change the value of the exponent by the shift in the calculation result of the mantissa.

(5) The output results with normalized processing. According to the IEEE-754, it is obtained that the result is standardized, and the output of a 32-bit result is ensured. Done_Sig is a high continuous output before 32-bit output.

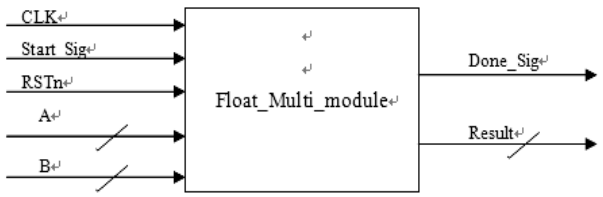

Figure 2: Floating-point Multiplier Module

(2) Floating-point Multiplier Module Implementation

A module of the establishment of floating-point multiplier is shown in Figure 2, where CLK is a clock signal. While Start_Sig module activate signals for the low-level reset signal, and A, B, RSTn input 32-bit single-precision floating-point number. Result is the output of the 32-bit calculation result and Done_Sig is the completion of the feedback signal. In order to ensure consistency feedback of calculation completion in neural network implementation of the finite element, the computer clock of the multiplier and adder 8 is set, and then Done_Sig outputs the high level information [Wang and Dai, 2009].

The following steps are required to achieve Floating point multiplier.

(1) The preprocessing of input operand. The standard format of the operand sign bits, the exponent bits and the mantissa bit have to be separated. In order to ensure the accuracy of the calculation result, the 23 bits "0" behind the mantissa, two "01" ahead of mantissa are filled, and two "00" is before the exponent.

(2) Operands operations. XOR operation of the sign bit, the exponent adder and the multiplication of the mantissa are used.

(3) Calculation results processing. The calculation result of the sign bit is the sign bit of the result. Ensure the first mantissa to be 1, according to the calculation result of the mantissa. The decimal point position is shifted on the basis of the shift to determine the order of code bits.

(4) The output results with normalized processing. According to the IEEE-754, it is obtained that the result is standardized, and the output of a 32-bit result is ensured. Done_Sig is a high continuous output before 32-bit output.

(3) The Module of Floating-point Multiplier-Adder Implementation

In the algorithm, the extensive use of A*B+C*D computing modules, and in order to ensure parallel computing, the mul_add_module.v module are designed and implemented.

This is connected with two floating point multiplier and a floating point adder. Operands which are inputs for the A, B, C, D in four 32-bit wide are operated by inputting to the multiplier module in pairwise respectively, and then parallel computing of multiplier is realized. After the output, the outputted procedure of the multipliers is used to complete the feedback signal Done_ Sigthe and operating as a module of the floating-point adder to start signal Start_Sig, the result of the multiplier as the operand input of the adder operation. The output result signal is Done_Sig. Module instantiated is shown in Figure 3.

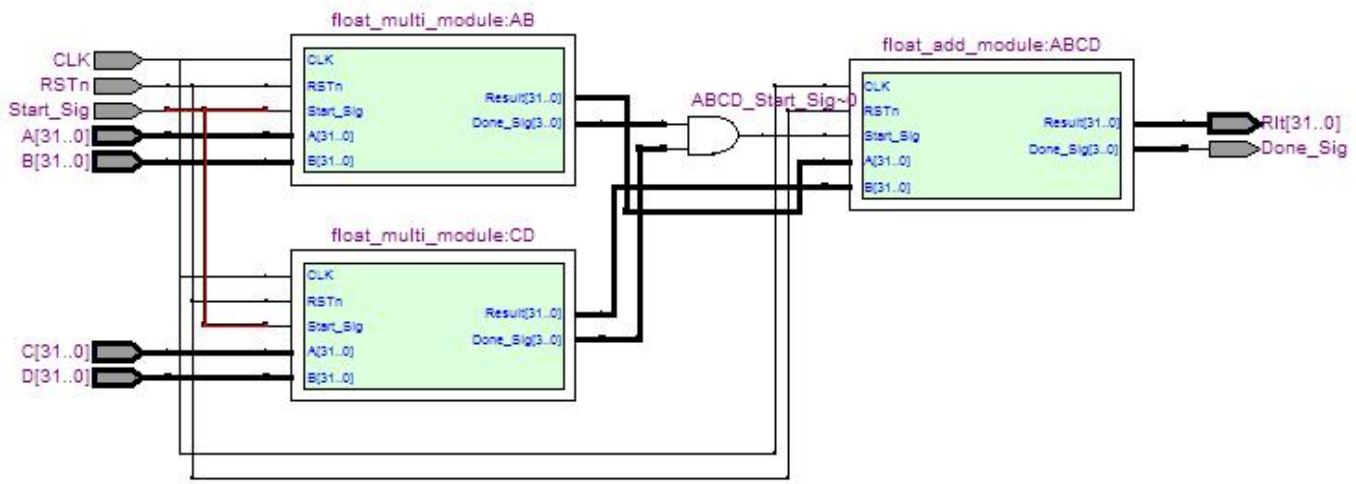

Figure 3: Implementation of Mul_add_module.v Module

Emulating Computation

Example 1 suppose that there is a triangle plate with thickness t = 0.1 mm; elastic modulus of the material E = 1000 Pa, the Poisson's ratio l = 0 and F = 10 KN. The dimension and the constrains are shown. The triangle plate is meshed into four triangular elements, totally six nodes in Figure 4. Without considering the effect of the gravity, the routine method of finite element and the computational method of the neural network of the finite element are adopted to solve the displacement of every node separately.

First, the element stiffness matrix of every triangular element are calculated, and then the general manipulative equations are obtained after combining the element, introducing the boundary conditions and correcting the general stiffness matrix.

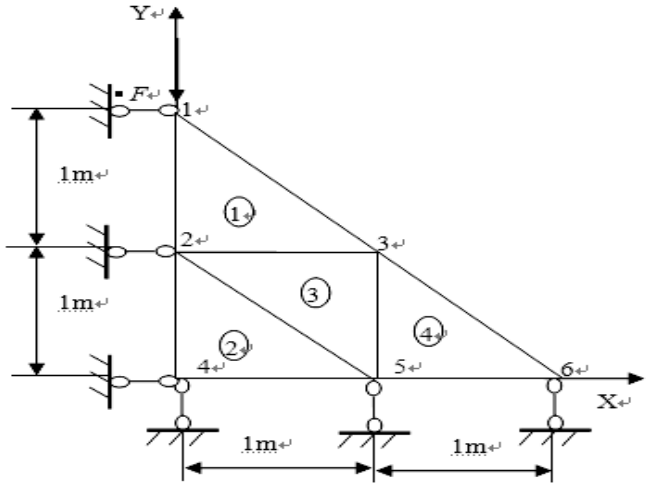

Figure 4: Loading and Finite Element of Triangle Sheet

$$\begin{bmatrix} 0.5 & -0.5 & 0.0 & 0.0 & 0.0 & 0.0 \\ -0.5 & 1.5 & -0.25 & -0.5 & 0.25 & 0.0 \\ 0.0 & -0.25 & 1.5 & 0.25 & -0.5 & 0.0 \\ 0.0 & -0.5 & 0.25 & 1.5 & -0.25 & 0.0 \\ 0.0 & 0.25 & -0.5 & -0.25 & 1.5 & -0.5 \\ 0.0 & 0.0 & 0.0 & 0.0 & -0.5 & 0.5 \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_3 \\ u_5 \\ u_6 \end{Bmatrix} = \begin{Bmatrix} -1.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{Bmatrix}$$

According to Eq. (3) and the construction of neural network as shown in Figure 5, the system of neural network could be obtained.

In Figure 5, u_i (variables to be obtained) is the output, and q_i (load vector) is the input. In one iteration, u_i are fed back parallel to computing elements laid in the ith row, and then 22 multiply operations are processed. Here, k_{ji} is an element in the ith row and the jth column of global stiffness matrix. After doing accumulation in six elements named 'f' (multiple input and single output), the results can be obtained.Z^{-1} is time delay [Zhou, *et al.*, 2005].

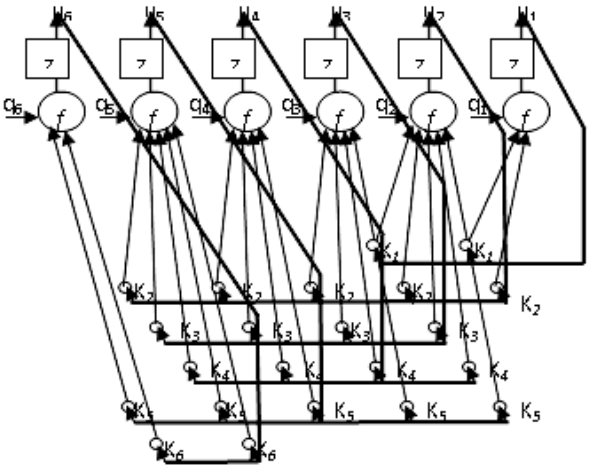

Figure 5: Neural Computation Block Diagram

First of all, the value of the step size should be considered. According to Table 2, the time step $\Delta t < 0.7803$. The value of step size can be calculated. Corresponding to the different Δt (the neuro-computation can be operated), after taking the origin of coordinate as the initial point of iterative computation, $\|e\|_2 < 0.01$ ('e'is the error) is taken as the condition of convergence.

Table 2: The Relationship Between Convergence Time (number of iterations) and Iterative Step Length (Δt).

Iterative Step(Δt)	0.1	0.3	2/3	0.75	Steepest Descent	0.80
Iterations	235	77	33	31	29	Divergent

And then, digital experiments could be conducted by choosing different initial points. Six groups of initial points are given to make them typical. The vectors, which are formed by initial points and origin of coordinates, are orthogonal. The six groups of initial points are:

x_1=(-3.3346, 5.3281, 0.1176, -0.7961, 5.4552, -5.4850)
x_2=(-2.7896, -3.1493, -7.2920, -5.1820, 1.3809, 0.6059)
x_3=(-5.0548, 3.2908, 0.2993, -1.8711, -7.6719, -1.0826)
x_4=(-4.6158, -5.4101, 6.2856, -2.6640, 1.6553, -0.2815)
x_5=(-3.6230, -3.7163, -2.6794, 7.4256, -0.6697, -3.2088)
x_6=(-4.5953, 2.7899, -0.1892, 2.6031, 2.5075, 7.6158)

With point of the six sets of orthogonal neural network being the initial input, and the time step $\Delta t = 2/3$, the neural network operation of step 100 of the results are listed in Table 3 by running the neural network.

Table 3: Iteration of Step 100 of Neural Network Output Values With the Theoretical Value

Initial points	u1	u2	u3	u4	u5	u6
x1	-3.2525	-1.2539	-0.0868	-0.3725	0.1747	0.1761
x2	-3.2525	-1.2536	-0.0871	-0.3728	0.1749	0.1760
x3	-3.2528	-1.2526	-0.0880	-0.3737	0.1760	0.1758
x4	-3.2529	-1.2522	-0.0884	-0.3741	0.1763	0.1757
x5	-3.2529	-1.2521	-0.0885	-0.3742	0.1765	0.1757
x6	-3.2527	-1.2529	-0.0877	-0.3734	0.1756	0.1759
Theo-retical value	-3.2527	-1.2527	-0.0879	-0.3736	0.1758	0.1758

Computation of FPGA

According to the calculation example, it is the calculation of FPGA design. The use of the basic modules float_a-dd_modu-le.v, float_multi_ module.v and mul_add_m-odule.v are designed. According to neural network structure in Figure 5, we design synthesis in the Quartus II, and carry out the functional simulation under the Modelsim. Results are shown in Figure 6.

where Counter_reg is the number of iterations; Sig_done_reg feedback is the signal after the completion of the one iteration; Sig_start_reg is a start signal for the next iterated calculation. Through the procedure of the normalization process of the result, intercepting the 23 mantissa, the precision of the floating point arithmetic has been improved by the law of Roundup. Because of the regulatory function of self-accuracy of the neural network computation, Table 4 shows that there are small errors between the FPGA simulation results and the theoretical values of calculations, but the range is acceptable.

Conclusions

In the implementation of parallel computing of finite element neural network, we achieved it by Stratix II EP2S130F1508C5 chip. After general layout, results show that combinational ALUTs are 55789, which account for 52% of the totality; dedicated logic registers for 16460, which account for 16% of the totality; DSP block 9-bit elements are 336, which account for 67% of the totality. When frequency of simulation is 250MHz, the length of completion of one iteration is 216ns. In the simulation, the serial computational method was adopted by the software of MATLAB. In this paper, parallel implementation is used by FPGA, which greatly improves the calculation speed to save the computing time. Meanwhile it guarantees the accuracy, and provides a new calculation method for improving neural network parallel finite element calculation.

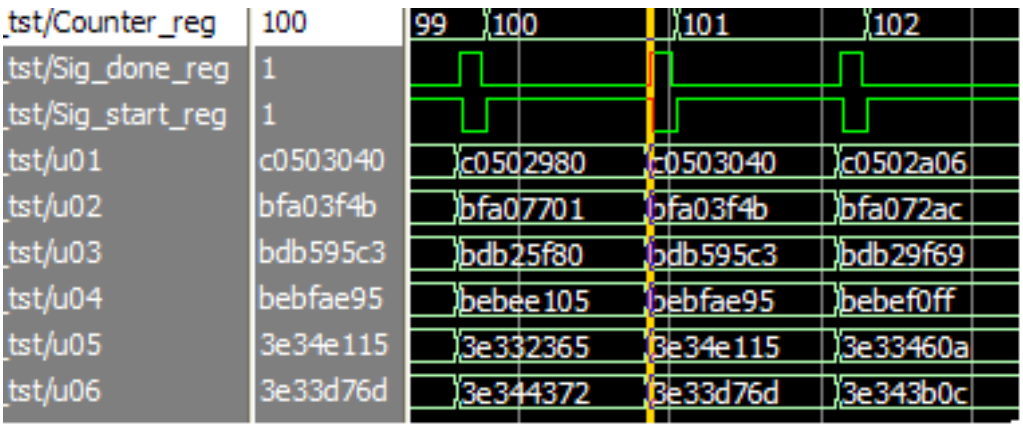

Figure 6: Output of FPGA Simulation

Table 4: The neural Network Simulation Results of Iteration of Step 100

Initial value	u1	u2	u3	u4	u5	u6
x	-3.2525	-1.2539	-0.0868	-0.3725	0.1747	0.1761
Theoretical value	-3.2527	-1.2527	-0.0879	-0.3736	0.1758	0.1758
FPGA-D	-3.25294	-1.25193	-0.0886646	-0.374379	0.17664	0.175626
FPGA-H	C0503040	BFA03f4b	BDB595C3	BEBFAE95	3E34E115	3E33D76D

References

Consolazio GR (2000) Iterative Equation Solver for Bridge Analysis using Neural Networks. *ComputerAided Civil and Infrastructure Engineering*, Eng., **15**: 107-119.

Gao HS; Zhang J; and **Qin WY** (2008) On Applying CG Network to Structural Analysis. *Proceedings, 4th International Conference on Natural Computation*, *ICNC*, 2, 18-20 Oct. 2008, Jinan, China, pp560-563.

IEEE-754 (2008) IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2008, *Microprocessor Standards Committee,* **26** (6): 1578-1581.

Li HB; Duan W; and **Huang HZ** (2010) Neurocomputing Method Based on Structural Finite Element Analysis of Discrete Model. *Neural Computing and Applications*, **19** (6): 875-882.

Li HB; Huang HZ (2004) Finite element Analysis of Structures Based on Linear Saturated System Model. *Lecture Notes in Computer Science*, **3174/2004**: 820-825.

Liu Y; Yin Arendt XP; Chen W; and **Huang HZ** (2010) A Hierarchical Statistical Sensitivity Analysis Method for Multilevel Systems with Shared Variables. *Journal of Mechanical Design,***132** (3): 031006-1-031006-11.

Liu Y; Zhou W; and **Liu F** (2005) Parallel Computing Analysis of Arch Dam and Foundation System Using 3-D FEM with Element by Element Approach. *Qinghua Daxue Xuebao/Journal of Tsinghua University*, **45** (6): 772-775.

Wang S; and **Dai YH** (2009) Single Precision Floating-Point Adder FPGA Realization. *Chinese Modern Electronic Technology*, **8** (1): 8-10.

Wang Z; Huang HZ; and **Liu Y** (2010) A Unified Framework for Integrated Optimization under Uncertainty. *Journal of Mechanical Design,***132** (5): 051008-1-051008-8.

Zhang X; Huang HZ; and **Xu H** (2010) Multidisciplinary Design Optimization with Discrete and Continuous Variables of Various Uncertainties. *Structural and Multidisciplinary Optimization*, **42** (4): 605-618.

Zhou N; Chen Y; and **Li A** (2005) Design and Implementation of Floating Point Calculator Based on FPGA Technology. *Chinese Computer Engineering and Design*, **26** (6): 1578-1585.