Parallel multi core back propagation neural network learning algorithm for simulation and prediction of cyclic loading of steel specimens

Agnieszka Krok

Department of Physics, Mathematics and Computer Science, Cracow University of Technology
ul. Warszawska 24, 31-155 Kraków, Poland
e-mail:agakrok@poczta.fm

Abstract

Two versions of batch mode back propagation neural networks learning algorithm were proposed. Both algorithms were executed on multi core parallel computer with shared memory. The effectiveness of both algorithms was tested on the task of modeling and prediction of cyclic loaded steel specimens. The tests based on experimental data for
AISL 316L stainless steel.

Keywords: cyclic loadings, neural networks, parallel computing, steel

1. Introduction

Artificial neural networks (ANN) were proved to be very effective tools for the analysis of structures and material problems, [4,7]. From among them the multi layer feed forward ANN learned by means of Back Propagation algorithms are the most frequently used. The growing dimensions and complication of the problems to be solved, led to the longer computational time. The answer to this problem might be the multi core parallel computers, which enable processing the different parts of the code in the same time, [6].

2. Parallel batch mode back propagation neural networks learning algorithm

There are several ways to rewrite existing serial code into the parallel version. All of them assume to separate the program into small task according to the Bernstain conditions, [1]. According to the Flynn's Taksonomy, [3], the proposed parallel models are in the form SIMD (single-instruction-multiple-data) instead of SISD (single-instruction-single-data) used in the classical versions of both algorithms. In the presented paper the OpenMP portable, for developing the parallel code was applied, [2], together wit C++ . It includes tools for creating tasks and threads, sharing the tasks among the threads, synchronization the tasks, data environment and runtime functions and variables. Two ways of decomposition was considered:

- input parallelism – the collection of learning patterns was distributed among tasks: into subsets processed independently, but in the SIMD schedule

- functional parallelism – independent parts of algorithms are computed in the same time, tasks are formulated either according to the code formulation – ie. parallelization of particular loops, or according to the ANN formulation ie. parallelization according to the neurons.

The task formulation had to be done very careful to avoid the data race, for the ANN with distributed information flow, and the sequence algorithm of minimalization of the error in the multidimensional parameter space.

The used OpenMP directive to share the work among the threads were:
pragma omp parallel for – the beginning of the parallel section within sequence code, parallelization of 'for' directive, the barrier set after this parallel section, when all the threads wait until the last one would do its share;
shared(variables) – pointing the variables which are known and can be modified by all the threads, one existing copy for all the threads;
private(variables) - each thread has his own copy of these variables, other threads can modify only their own copies;
num_threads(number) – setting up the number of threads;
reduction(operator : variable) – pointing out the particular operator applied for collection of local copies of variable for each thread, gathering them into one variable giving out of parallel section to the sequence part of the code

Parallel batch mode back propagation neural networks learning algorithm assumed correction of ANN weights after presentation of all the learning patterns ie. ones for one epoch in the batch mode. The input parallelism was applied in the following form: the collection of patterns was divided among all the threads. Equal share was assumed. The single portion of work includes calculation of the averaged error for the set of patterns assigned to the particular threads. The correction of weights was made in the serial mode, because in the comparison of calculating averaged error this is negligible as far as execution time is concerned**.** Functional parallelism was applied to spread the work within the calculation loops in the weight updating algorithm.

The parallel computer SGI Altix ICE 8200, placed in the Institute of Computer Science at Cracow Institute of Technology. The computer consists of 32 main servers and 3 additional helping servers. Each unit consists of 2 processors and operating memory 16 GB, with 800 MHz clock rate. Each processor is the is the Intel Quad Core Xeon 5472, 3GHz, 1600 MHz FSB, with local memory 2x6MB, 48 GFlops of theoretical peak performance. Source code was written in c++ with OpenMP, compiled using Intel ICC Compiler and executed in Unix environment.

Table 1: Numerical results in CPU clock tics

	Small ANN, x~5	Large ANN, x~80
	CPU time	CPU time
1 thread	569850000	2505300000
2 threads	45980000	196360000
3 threads	35620000	143550000
4 threads	24710000	92700000
5 threads	7800000	7302000
6 threads	45590000	757000

3. Numerical results

Numerical test were proceeded for 3-N-1 ANN, where N varies from 10 to 100 hidden neurons. Simulation of hysteresis loops for steel specimens are made for discretized experimental data. The first nine loops containing L = 9×49 patterns for the k = 1,.., 441 were used for the learning and T = 588– 441 = 147 = 3×49 patterns form final loops for k = 442,..,588 for testing. Input vector x consist of scaled marker of current pattern k/587, scaled marker of current pattern number inside each loop separately: mod(k,49)/49 and the previous σ given by 3-N-1 ANN: σ (ssn)(k-1), [5], see Fig.1:

$$x(k)=[\sigma(ssn)(k-1),\ k/587,\ \mathrm{mod}(k,49)/49] \quad (1)$$

The Back Propagation with momentum term algorithm is by the set of equations:

$$\boldsymbol{w}(t+1)=\boldsymbol{w}(t)+\Delta\boldsymbol{w}(t) \quad (2)$$

where

$$\Delta\boldsymbol{w}(t)=\eta \ \text{grad}\ E(\boldsymbol{w}(t))+\ \alpha\Delta\boldsymbol{w}(t-1) \quad (3)$$

where η – the learning coefficient, assumed 0.25; α – the coefficient of the share of the previous weight update in the current weight update, assumed 0.9 (t=0,1,2...)

To prevent the over fitting the both algorithms are examined on the growing of TestErrors:

1.learn the network for 10 epochs
2. if TestError < MinTestError, change weight matrix, if not don't change
3. if TestError > 1.2 * MinTestError stop training

The effectiveness of the solution was examined as far as the number of threads are concerned for two groups of neural networks: small (about 5 hidden neurons) and large (up to 100 hidden neurons).

4. Conclusions

Parallel batch mode back propagation neural networks learning algorithm can be successfully parallelized using OpenMp environment. Input parallelism or functional parallelism can be used. The time of ANN learning can be shorten using dividing the calculating of ANN error for all learning patterns into several threads and calculating the independent parts of weight updating procedures in the same time as the parallel processes. The speed up of the learning process is larger when bigger ANN are necessary to solve given numerical problem.

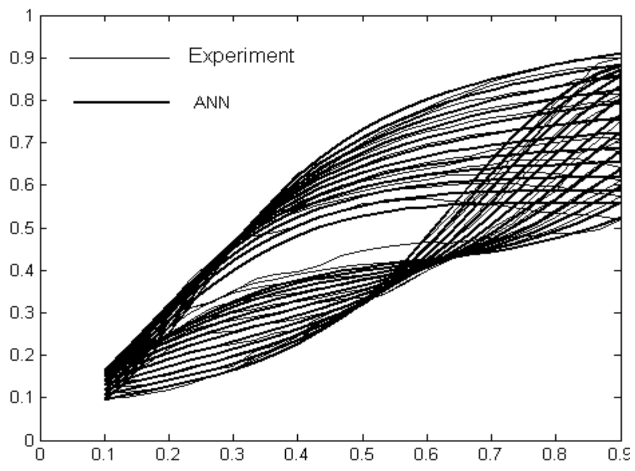

Figure 1: Plot of a stress-strain curve for AISL 316L stainless steel

References

[1] A.J. Bernstein, *Program Analysis for Parallel Processing*, IEEE Trans. on Electronic Computers, EC-15, 1966, s. 757–762

[2] B.Chapman, G.Jost, R van der Pas, *Using OpenMP portable shared memory parallel programming*, MIT press, 2008

[3] M. Flynn, Some Computer Orgamzations and Their Effectiveness, IEEE Trans. Comput. Vol C-21, 1972.

[4] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1998

[5] J. Lemaitre, *A course on damage machanics*, Springer, 1992

[6] B.H.V.Topping, J.Sziveri, A.Bahreinejad, J.P.B. Lite, B.Cheng, *Parallel processing, neural networks and genetic algorithms, Advs.in Eng. Soft.*, vol. 29, no. 10, pp.763-786, Elsevier, 1998.

[7] Z. Waszczyszyn, *Neural Networks In Analysys and design of structures*, Springer, 1999.