# Modeling Resonant Frequency of Rectangular Microstrip Antenna Using CUDA-Based Artificial Neural Network Trained by Particle Swarm Optimization Algorithm

**Feng Chen and Yu-bo Tian**

School of Electronics and Information
Jiangsu University of Science and Technology, Zhenjiang 212003, Jiangsu, P. R. China
cfcfchenfeng@163.com, tianyubo@just.edu.cn

***Abstract* ─** Resonant frequency is a vital parameter in designing Microstrip Antenna (MSA). Artificial Neural Network (ANN) based on Particle Swarm Optimization (PSO) algorithm (PSO-ANN) has been used to model the resonant frequency of rectangular MSA. To deal with the problem of the long execution time when training PSO-ANN, its parallel implementation in the Graphic Processing Unit (GPU) environment is proposed in this paper. The presented approach uses the particle behavior parallelization of PSO to accelerate ANN training, and is applied to modeling the resonant frequency of rectangular MSA under Compute Unified Device Architecture (CUDA). Experimental results indicate that compared with CPU-based sequential PSO-ANN, more than 300 times of speedup ratio has achieved in GPU-based parallel PSO-ANN with the same optimization stability. Furthermore, the network error can be significantly reduced with the very limited runtime increment when substantially enlarging the number of particles on GPU side.

***Index Terms* ─** Artificial Neural Network (ANN), Compute Unified Device Architecture (CUDA), Microstrip Antenna (MSA), Particle Swarm Optimization (PSO), resonant frequency.

## I. INTRODUCTION

Microstrip Antenna (MSA) is used in a broad range of applications in communication systems, and this is primarily due to its thin profile, small size, light weight, and low manufacturing cost [1,2]. As is known to all, MSA has narrow frequency band and works effectively only in the vicinity of its corresponding resonant frequency, which is a vital parameter in designing MSA. So a model to determine the resonant frequency is helpful in antenna design. Many scholars have proposed some traditional methods with different accuracy and computing power to calculate the resonant frequency of the most commonly used rectangular MSA [3-13].

In the past several years, Artificial Neural Network (ANN) model has been used in antenna design, including modeling the resonant frequency of rectangular MSA [14,15] due to its excellent abilities of learning and generalization, little memory requirement and fast real-time operation. The related data of the antenna can be got by measurement or simulation. After training these data, the ANN related to the antenna design problem can be achieved, and this quickly provides solutions to the problem. Particle Swarm Optimization (PSO) algorithm [16,17] has been gradually applied to ANN training (PSO-ANN) due to its simple concept, easy implementation, and strong abilities of convergence and global search. PSO-ANN has been used to model the resonant frequency of rectangular MSA and proved with better convergence precision and stronger predictive ability than common BP-based ANN (BP-ANN) [18-20]. However, PSO-ANN needs long computing time, especially for large scale problems, such as the problem of modeling the resonant frequency of rectangular MSA. Parallel optimization is an effective way to solve this problem.

Besides ANN's data parallelization and node parallelization [21], PSO's natural particle

*Submitted On: February 27, 2014*
*Accepted On: September 24, 2014*

behavior parallelization is in PSO-ANN. There are many parallel ways to accelerate PSO algorithm. Compared with computer cluster [22,23], multi-core CPU [24] or other professional parallel devices like FPGA [25,26], graphic processing unit (GPU) [27,28] has the most significant advantages in hardware cost. Since the NVIDIA company introduced the Compute Unified Device Architecture (CUDA) in 2007, CUDA has become the most popular GPU programming architecture due to its excellent programmability.

Ground on the existing research of GPU-based PSO algorithm, we design the CUDA-based parallel PSO-ANN scheme to fast model the resonant frequency of rectangular MSA in this paper. Experimental results show that when using the same number of particles on GPU side, the modeling runtime can be greatly reduced and more than 300 times of speedup ratio has obtained, while the modeling error is similar or same to the CPU-based program. When using substantially more number of particles on GPU side, the modeling error can be significantly reduced and better than the corresponding results in literatures.

The rest of this paper is organized as follows. Section II briefly discusses the calculation formula of the resonant frequency of rectangular MSA. Section III slightly introduces the implementation of PSO-ANN algorithm on CPU side. The CUDA-based parallel implementation of PSO-ANN is presented in Section IV. We use the GPU-based parallel PSO-ANN to rapidly model the resonant frequency of rectangular MSA, give the performance results, and provide some employment suggestions in Section V. Some concluding remarks of this work are finally reported in Section VI.

## II. RESONANT FREQUENCY OF RECTANGULAR MSA

The model of rectangular MSA is shown in Fig. 1. Its length, width, dielectric substrate's thickness, and relative dielectric constant are $W$, $L$, $h$ and $\varepsilon_r$, respectively. The resonant frequency of rectangular MSA can be calculated as formulas (1)~(3) [1,2]:

$$f_{mn} = \frac{c}{2\sqrt{\varepsilon_e}}\left[\left(\frac{m}{L_e}\right)^2 + \left(\frac{n}{W_e}\right)^2\right]^{1/2}, \qquad (1)$$

where $\varepsilon_e$ is the effective relative permittivity. $c$ is electromagnetic wave propagation velocity in vacuum. $m$ and $n$ are integers. $L_e$ and $W_e$ are effective length and width. When calculating the resonant frequency of rectangular MSA in main mode TM10, formula (1) can be written as:

$$f_{10} = \frac{c}{2L_e\sqrt{\varepsilon_e}}\,. \qquad (2)$$

The effective length can be defined as follows:

$$L_e = L + 2\Delta L\,, \qquad (3)$$

where $\Delta L$ is the boundary extension length, which is connected with dielectric substrate's thickness $h$.

Obviously, the resonant frequency of rectangular MSA depends on $h$, $\varepsilon_r$, $m$, $n$, $W$ and $L$.

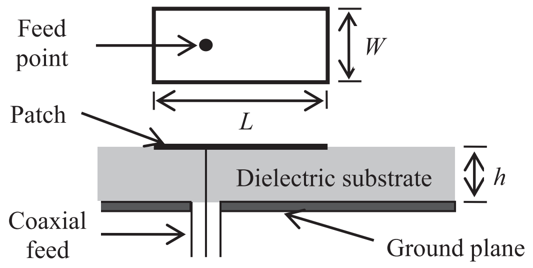

Fig. 1. Model of rectangular MSA.

## III. PSO-BASED ANN

### A. Standard PSO algorithm

There have been many versions of PSO algorithm. The version introducing inertia weight [16] is used and called "Standard PSO" in this paper. The optimization problem dimension is $D$ and the number of particles is $N$. The positions of each particle represent a potential solution to the problem in the $D$-dimensional search space, and the velocities of each particle represent its movement. All particles have fitness values that are evaluated by the fitness function to be optimized. During each of the iteration, the positions and velocities of every particle are updated according to its Personal best positions (Pbest) and the Global best positions (Gbest). The velocities and positions updating in PSO can be formulated as follows:

$$V_{id}(t+1)=wV_{id}(t)+c_1r_1(Pbest_{id}(t)-X_{id}(t)) +c_2r_2(Gbest_d(t)-X_{id}(t)), \qquad (4)$$

$$X_{id}(t+1)=X_{id}(t)+V_{id}(t+1). \qquad (5)$$

In equation (4) and equation (5), $i$=1,2,...,$N$ and $d$=1,2,...,$D$. The learning factors $c_1$ and $c_2$ are nonnegative constants. $r_1$ and $r_2$ are random numbers uniformly distributed in [0,1]. $V_{id}(t)\in[-V_{\max},V_{\max}]$, where $V_{\max}$ limits the maximum velocity of each dimension of the particle. $X_{id}(t)\in[-X_{\max},X_{\max}]$, where $X_{\max}$ limits the maximum position of each dimension of the particle. Usually $V_{\max}=kX_{\max}$, where $0\leqslant k\leqslant 1$. The inertia weight $w$ is used to balance the ability between global exploration and local exploitation, and can be either a constant or a variable in [0,1].

## B. ANN trained by PSO algorithm

PSO algorithm can be used to train ANN. ANN training includes optimization of ANN's structure and optimization of ANN's weights and thresholds (hereinafter referred to as weights). This article only concerns the optimization of ANN's weights under the condition of the given ANN's structure. ANN's weights must be encoded before training. There are two encoding strategies, namely vector encoding and matrix encoding. Vector encoding is chosen in this paper. For convenience, a feedforward ANN with 2 nodes in the input layer, 3 nodes in the hidden layer, and 1 nodes in the output layer (13-dimensional), is shown in Fig. 2.

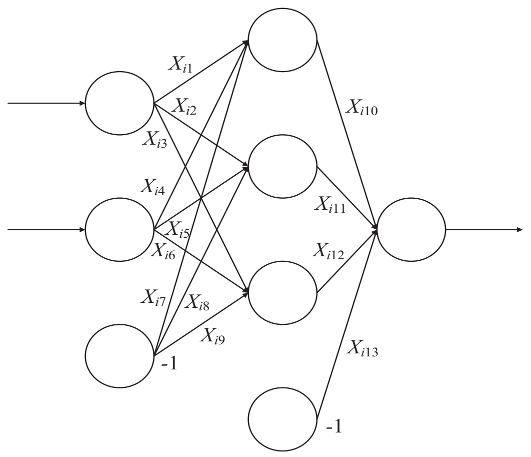

Fig. 2. Feedforward ANN model with 2-3-1 structure.

In PSO-ANN, each particle is encoded to a vector, representing a solution of ANN's weights [18]:

$$\boldsymbol{P}_i=[X_{i1}\,X_{i2}\,X_{i3}\cdots X_{i11}\,X_{i12}\,X_{i13}]. \tag{6}$$

Each particle's fitness value is set as its corresponding Mean Squared Error (MSE) of ANN's output of training samples. All the particles revise their values by PSO algorithm. The final best solution Gbest is namely the well-trained ANN's weights. It is easy to see that PSO-ANN is essentially the special PSO algorithm whose fitness value is ANN's output error.

The steps of PSO-ANN algorithm used in this paper are as follows:

(a) Load training samples and testing samples. Data preprocesses. Set the maximum iteration number $T_{\max}$.
(b) Initialize all particles' positions $X_{id}(t)$ and velocities $V_{id}(t)$ at random.
(c) Initialize all personal best positions $Pbest_{id}(t)$ and the global best positions $Gbest_d(t)$.
(d) Update all particles' velocities $V_{id}(t)$ and positions $X_{id}(t)$ according to equation (4) and equation (5).
(e) Evaluate all particles' fitness values F($X_i$).
(f) Update all personal best positions $Pbest_{id}(t)$ and their corresponding fitness values F($Pbest_i$). Update the global best positions $Gbest_d(t)$ and their corresponding fitness value F($Gbest$).
(g) If the iteration number reaches $T_{\max}$, go to step (h), else go to step (d).
(h) Evaluate total output error of training samples and testing samples.

## IV. CUDA IMPLEMENTATION OF PARALLEL PSO-ANN

### A. CUDA programming model

CUDA adopts the CPU+GPU heterogeneous cooperative computing platform. As the host, CPU takes responsibility for logic processing and serial computing. As the device or coprocessor, GPU takes responsibility for compute-intensive, highly parallel computing. CUDA uses similar C language as its basic programming language to achieve good programmability and portability. A CUDA kernel is a parallel function, which follows the SIMT (Single Instruction, Multiple Threads) execution model on GPU. CUDA program process typically includes the following 6 steps:

(1) Allocate and initialize CPU memory.
(2) Allocate GPU memory.
(3) Transfer data from CPU side to GPU side.
(4) Perform parallel computing on GPU side.

(5) Transfer results from GPU side back to CPU side.
(6) Process data obtained in step (5) on CPU side.

## B. Design scheme and specific realization

Currently, the parallel implementation of ANN training mainly uses two parallelization strategies, namely data parallelization and node parallelization. However, for a common ANN, the number of neuron nodes or training samples is often only from ten to several ten. Therefore, these two strategies are suitable for parallel ways like computer cluster [21,29], but somewhat unsuitable for GPU because they have not enough parallel degree. Besides ANN's data parallelization and node parallelization, PSO's natural particle behavior parallelization is in PSO-ANN. For some complex problems, the number of particles can be from hundred to several hundred or more. Therefore, particle behavior parallelization is quite suitable for GPU architecture, which needs as many threads as possible to make full use of its powerful parallel computing ability.

In 2009, Veronese and Krohling firstly used CUDA to accelerate the PSO algorithm [27], which raised the research upsurge in GPU-based parallel PSO algorithm [28]. Particle behavior parallelization in PSO-ANN can be reflected in three aspects: (a), (b), and (c), as follows. In addition, CUDA-based PSO-ANN can use CUDA's unique parallelism, which can be reflected in aspect (d).

(a) The process of updating particles' velocities and positions is parallel.
(b) The process of evaluating particles' fitness values is parallel.
(c) The process of updating personal best positions and their corresponding fitness values is parallel.
(d) CUDA's parallel reduction algorithm can accelerate the process of finding the minimum fitness value when updating the global best positions.

According to the analyses above, the approach of GPU-based parallel PSO-ANN algorithm is designed in Fig. 3. The proposed approach corresponds one particle to one thread, and deals with a large number of GPU threads in parallel. This greatly saves the computing time and improves the computing accuracy.

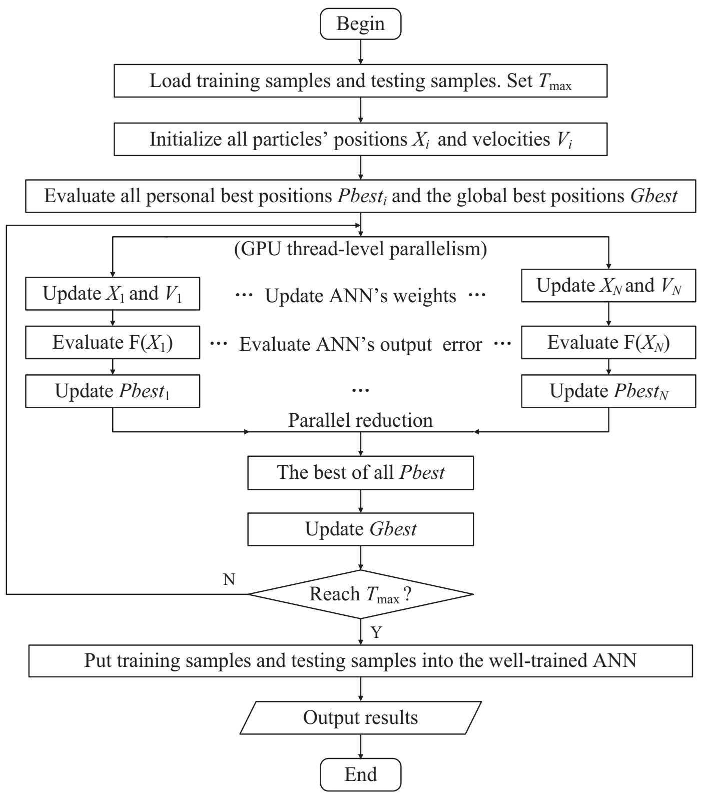
Begin
Load training samples and testing samples. Set $T_{max}$
Initialize all particles' positions $X_i$ and velocities $V_i$
Evaluate all personal best positions $Pbest_i$ and the global best positions $Gbest$
(GPU thread-level parallelism)
Update $X_1$ and $V_1$
··· Update ANN's weights ···
Update $X_N$ and $V_N$
Evaluate F($X_1$)
··· Evaluate ANN's output error ···
Evaluate F($X_N$)
Update $Pbest_1$
···
Update $Pbest_N$
Parallel reduction
The best of all $Pbest$
Update $Gbest$
N
Reach $T_{max}$?
Y
Put training samples and testing samples into the well-trained ANN
Output results
End

Fig. 3. CUDA-based parallel PSO-ANN algorithm flowchart.

The steps of GPU-based parallel PSO-ANN algorithm are as follows:

(1) Load the training samples and testing samples on CPU side. Data preprocesses.
(2) Call the malloc() function and cudaMalloc() function on CPU side. Allocate variable space on CPU side and GPU side.
(3) Initialize particles' positions and velocities on CPU side.
(4) Call the cudaMemcpy() function on CPU side, and transfer the data of particles from CPU side to GPU global memory. Call the cudaMemcpyToSymbol() function on CPU side, and transfer training samples from CPU side to GPU constant memory.
(5) Call the kernel() function on CPU side. Perform parallel computing tasks (ANN training) on GPU side.
(6) Call the cudaMemcpy() function on CPU side, and transfer the useful data (well-trained ANN) from GPU side back to CPU side.
(7) Evaluate the output results on CPU side by training samples, testing samples and well-trained ANN.
(8) Call the free() function and cudaFree() function on CPU side. Release variable space on CPU side and GPU side.

The step (5) is used to accelerate ANN training, and it is the core step of the whole algorithm. The pseudo-code of step (5) is as follows:

```
for (i=0; i<generationsNumber; i++)
{
<Update velocities and positions of each particle>
                                                // kernel 1
<Compute fitness of each particle>              // kernel 2
<Update Pbest of each particle>                 // kernel 3
<Update Gbest of all particles>                 // kernel 4
}
```

## V. MODELING RESONANT FREQUENCY OF RECTANGULAR MSA USING CUDA-BASED PARALLEL ANN-PSO

In this section, we use CPU-based sequential PSO-ANN and the designed GPU-based parallel PSO-ANN, respectively, to model the resonant frequency of rectangular MSA and test their acceleration performance. The computing platform used in our experiments is shown in Table 1. The input sets of samples $(W, L, h, \varepsilon_r)$ are the related parameters of rectangular MSA. The output set of samples ($f_{\mathrm{ME}}$) is the corresponding measured resonant frequency. The well-trained ANN can establish the mapping between the related parameters of the rectangular MSA and its corresponding measured resonant frequency. The training samples and testing samples used in this paper are from previous works [12,30]. Column 1-6 of Table 2 gives the total 33 sets of data, in which 26 sets of data are used for ANN training and the remaining 7 sets of data marked with asterisks are used for ANN testing. Column 2-5 of Table 2 shows the related parameters of rectangular MSA. Column 6 ("Theoretical $f_{\mathrm{ME}}$") of Table 2 shows the actual measured resonant frequency of rectangular MSA in mode TM10 ("theoretical values"). Tables 3 and 4 give the sum of the absolute error between experimental and theoretical values of the resonant frequency from traditional methods and CPU-based ANN models in different literatures. The $f_{\mathrm{EDBD}}$, $f_{\mathrm{DBD}}$, $f_{\mathrm{PTS}}$, $f_{\mathrm{PSO\text{-}BP}}$ and $f_{\mathrm{BiPSO}}$ in Table 4 represent, respectively, the experimental resonant frequency calculated by using the ANN model trained by EDBD (Extended Delta-Bar-Delta), DBD (Delta-Bar-Delta), PTS (Parallel Tabu Search), PSO-BP (PSO and BP together), and BiPSO (Binary PSO). It's worth noting that the "theoretical values" mean the actual measured resonant frequency (Column 6 of Table 2), while the "experimental values" mean the experimental resonant frequency from traditional methods [3-13], CPU-based ANN models [14-15,18-20], or our GPU-based parallel PSO-ANN model.

Table 1: Computing platform

| Name | Type |
|---|---|
| CPU | Intel Core i3-2100, 3.1 GHz |
| GPU | NVIDIA Tesla K20c, 706 MHz, 2496 CUDA Cores, Compute Capability 3.5 |
| Operating System | Windows 7 SP1 32 bit Professional |
| Programming Environment | Microsoft Visual C++ 2010, CUDA 5.0 |

Table 2: Experimental values of the resonant frequency of rectangular MSA in mode TM10 from the GPU-based PSO-ANN model

| No | $W$/cm | $L$/cm | $h$/cm | $\varepsilon_r$ | Theoretical $f_{ME}$ | Experimental $f_{ANN}$ | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.850 | 1.290 | 0.017 | 2.22 | 7740 | 7827.1 | 7775.0 | 7833.3 | 7756.6 | 7764.5 | 7764.0 | 7741.8 | 7763.0 | 7748.0 | 7764.9 | 7738.7 |
| 2* | 0.790 | 1.185 | 0.017 | 2.22 | 8450 | 8042.6 | 8063.7 | 8255.9 | 8163.5 | 8144.4 | 8170.8 | 8162.9 | 8126.0 | 8117.5 | 8170.3 | 8149.8 |
| 3 | 2.000 | 2.500 | 0.079 | 2.22 | 3970 | 3962.7 | 3949.1 | 3991.3 | 3988.6 | 3958.7 | 3974.0 | 3973.5 | 3969.4 | 3973.6 | 3964.6 | 3968.8 |
| 4 | 1.063 | 1.183 | 0.079 | 2.25 | 7730 | 7652.3 | 7678.8 | 7720.3 | 7664.3 | 7737.2 | 7706.0 | 7727.8 | 7712.2 | 7719.5 | 7701.5 | 7716.3 |
| 5 | 0.910 | 1.000 | 0.127 | 10.20 | 4600 | 4645.1 | 4592.5 | 4602.4 | 4607.6 | 4603.4 | 4598.3 | 4601.1 | 4598.1 | 4599.7 | 4601.8 | 4600.8 |
| 6 | 1.720 | 1.860 | 0.157 | 2.33 | 5060 | 5138.7 | 4943.5 | 5051.2 | 5048.8 | 5091.8 | 5042.6 | 5047.5 | 5040.9 | 5053.1 | 5061.2 | 5068.5 |
| 7* | 1.810 | 1.960 | 0.157 | 2.33 | 4805 | 4936.7 | 4663.9 | 4813.4 | 4767.8 | 4851.5 | 4774.3 | 4756.3 | 4758.5 | 4771.8 | 4812.6 | 4802.0 |
| 8 | 1.270 | 1.350 | 0.163 | 2.55 | 6560 | 6526.8 | 6568.4 | 6491.6 | 6524.5 | 6532.6 | 6527.1 | 6546.2 | 6551.0 | 6565.9 | 6537.1 | 6560.7 |
| 9 | 1.500 | 1.621 | 0.163 | 2.55 | 5600 | 5595.2 | 5588.1 | 5488.6 | 5611.1 | 5614.5 | 5629.5 | 5623.9 | 5629.6 | 5608.4 | 5626.7 | 5592.1 |
| 10* | 1.337 | 1.412 | 0.200 | 2.55 | 6200 | 6162.4 | 6205.3 | 6173.5 | 6187.2 | 6176.5 | 6181.1 | 6201.6 | 6190.4 | 6203.1 | 6180.7 | 6211.5 |
| 11 | 1.120 | 1.200 | 0.242 | 2.55 | 7050 | 7053.1 | 7138.2 | 7061.9 | 7075.9 | 7039.5 | 7064.1 | 7044.8 | 7054.6 | 7049.2 | 7062.6 | 7052.6 |
| 12 | 1.403 | 1.485 | 0.252 | 2.55 | 5800 | 5765.3 | 5809.7 | 5830.9 | 5805.4 | 5780.2 | 5804.5 | 5809.0 | 5798.4 | 5797.6 | 5792.4 | 5809.7 |
| 13 | 1.530 | 1.630 | 0.300 | 2.50 | 5270 | 5262.9 | 5277.2 | 5364.6 | 5248.4 | 5256.7 | 5268.4 | 5259.9 | 5264.0 | 5262.7 | 5265.8 | 5261.1 |
| 14 | 0.905 | 1.018 | 0.300 | 2.50 | 7990 | 7935.1 | 7935.3 | 7933.2 | 7978.9 | 8024.8 | 8032.1 | 8016.1 | 8024.3 | 8011.9 | 8022.2 | 8002.5 |
| 15 | 1.170 | 1.280 | 0.300 | 2.50 | 6570 | 6616.4 | 6661.0 | 6631.4 | 6595.5 | 6568.7 | 6560.3 | 6573.5 | 6560.3 | 6569.2 | 6553.5 | 6569.4 |
| 16* | 1.375 | 1.580 | 0.476 | 2.55 | 5100 | 5207.2 | 5155.0 | 5077.6 | 5112.1 | 5181.9 | 5177.8 | 5143.6 | 5227.8 | 5158.7 | 5083.9 | 5151.3 |
| 17 | 0.776 | 1.080 | 0.330 | 2.55 | 8000 | 7831.5 | 7880.8 | 7900.6 | 7948.2 | 7943.0 | 7959.1 | 7959.3 | 7965.7 | 7962.4 | 7961.5 | 7960.7 |
| 18 | 0.790 | 1.255 | 0.400 | 2.55 | 7134 | 7181.4 | 7183.7 | 7139.2 | 7183.0 | 7140.4 | 7130.7 | 7152.6 | 7144.4 | 7156.8 | 7145.4 | 7158.1 |
| 19 | 0.987 | 1.450 | 0.450 | 2.55 | 6070 | 6154.8 | 6066.4 | 6063.4 | 6108.5 | 6091.4 | 6108.2 | 6061.2 | 6062.3 | 6075.7 | 6085.4 | 6077.0 |
| 20* | 1.000 | 1.520 | 0.476 | 2.55 | 5820 | 5903.8 | 5835.7 | 5833.0 | 5880.7 | 5870.0 | 5894.9 | 5842.6 | 5853.4 | 5860.2 | 5869.6 | 5862.6 |
| 21 | 0.814 | 1.440 | 0.476 | 2.55 | 6380 | 6481.9 | 6377.8 | 6412.7 | 6395.0 | 6409.4 | 6392.9 | 6399.3 | 6393.7 | 6391.3 | 6395.6 | 6395.8 |
| 22 | 0.790 | 1.620 | 0.550 | 2.55 | 5990 | 5976.1 | 5958.7 | 5980.6 | 5895.0 | 5954.4 | 5955.1 | 5957.0 | 5969.3 | 5937.9 | 5950.8 | 5941.5 |
| 23 | 1.200 | 1.970 | 0.626 | 2.55 | 4660 | 4561.9 | 4643.9 | 4629.7 | 4619.1 | 4649.0 | 4650.7 | 4666.5 | 4694.3 | 4679.0 | 4660.8 | 4660.5 |
| 24 | 0.783 | 2.300 | 0.854 | 2.55 | 4600 | 4571.8 | 4734.5 | 4665.5 | 4629.1 | 4631.7 | 4636.6 | 4633.1 | 4611.1 | 4647.5 | 4636.5 | 4646.9 |
| 25* | 1.256 | 2.756 | 0.952 | 2.55 | 3580 | 3566.3 | 3623.1 | 3645.7 | 3508.0 | 3633.7 | 3538.3 | 3688.3 | 3583.9 | 3542.3 | 3558.2 | 3562.2 |
| 26 | 0.974 | 2.620 | 0.952 | 2.55 | 3980 | 3902.1 | 3990.6 | 3978.4 | 4021.5 | 3973.8 | 3962.6 | 3962.0 | 3958.3 | 3956.1 | 3970.1 | 3962.4 |
| 27 | 1.020 | 2.640 | 0.952 | 2.55 | 3900 | 3836.0 | 3920.4 | 3910.0 | 3947.0 | 3913.7 | 3889.6 | 3913.4 | 3895.8 | 3883.5 | 3898.3 | 3891.3 |
| 28 | 0.883 | 2.676 | 1.000 | 2.55 | 3980 | 3940.0 | 3990.9 | 3982.9 | 4021.0 | 3986.3 | 3984.2 | 3946.5 | 3973.4 | 3977.9 | 3985.9 | 3982.1 |
| 29 | 0.777 | 2.835 | 1.100 | 2.55 | 3900 | 3890.8 | 3856.1 | 3875.9 | 3857.0 | 3877.3 | 3884.8 | 3891.4 | 3902.5 | 3899.0 | 3886.8 | 3892.7 |
| 30 | 0.920 | 3.130 | 1.200 | 2.55 | 3470 | 3485.2 | 3433.9 | 3410.2 | 3455.6 | 3450.3 | 3463.3 | 3480.4 | 3476.4 | 3465.7 | 3461.2 | 3455.3 |
| 31* | 1.030 | 3.380 | 1.281 | 2.55 | 3200 | 3240.4 | 3197.1 | 3158.1 | 3184.1 | 3166.2 | 3198.6 | 3213.9 | 3196.4 | 3203.9 | 3193.7 | 3193.0 |
| 32 | 1.265 | 3.500 | 1.281 | 2.55 | 2980 | 3071.5 | 3042.3 | 3019.7 | 2956.3 | 3021.5 | 3005.2 | 2986.9 | 2983.3 | 2985.0 | 2990.9 | 3009.0 |
| 33 | 1.080 | 3.400 | 1.281 | 2.55 | 3150 | 3203.5 | 3159.9 | 3119.5 | 3140.3 | 3139.3 | 3156.5 | 3157.5 | 3152.0 | 3153.7 | 3148.9 | 3151.3 |
| Sum of the absolute error | | | | | | 2196 | 1702 | 1361 | 1293 | 1108 | 992 | 897 | 885 | 839 | 794 | 765 |
| Number of particles | | | | | | 128 | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 | 32768 | 65536 | 131072 |

Data with * are the sets of testing samples. The unit of frequency in this table is MHz.

Table 3: Sum of the absolute error between experimental and theoretical values of the resonant frequency of rectangular MSA in mode TM10 from traditional methods

| Traditional Method | Sum of the Absolute Error/MHz |
|---|---|
| [3] | 13136 |
| [4] | 24097 |
| [5] | 11539 |
| [6] | 12322 |
| [7] | 30996 |
| [8] | 8468 |
| [9] | 22572 |
| [10] | 18148 |
| [11] | 30504 |
| [12] | 56698 |
| [13] | 1393 |

Table 4: Sum of the absolute error between experimental and theoretical values of the resonant frequency of rectangular MSA in mode TM10 from CPU-based ANN models

| ANN Model | Sum of the Absolute Error/MHz |
|---|---|
| $f_{\mathrm{EDBD}}$ [14] | 2392 |
| $f_{\mathrm{DBD}}$ [14] | 2427 |
| $f_{\mathrm{BP}}$ [14] | 2372 |
| $f_{\mathrm{PTS}}$ [15] | 2239 |
| $f_{\mathrm{PSO}}$ [18] | 1049 |
| $f_{\mathrm{PSO\text{-}BP}}$ [19] | 1777 |
| $f_{\mathrm{BiPSO}}$ [20] | 863 |

In our experiment, the structure of ANN is designed as 4-10-1 and its corresponding particle dimension is 61. The number of particles is equal to the number of threads and is generally more than particle dimension (61 in our experiment). A warp is a group of 32 neighboring threads executed physically in parallel on a Stream Multiprocessor (SM) in CUDA. Therefore, the number of particles is designed to the multiples of 32. The activation function in the hidden layer is chosen as Bi-polar sigmoid function (formula (7)). The activation function in the output layer is chosen as Uni-polar sigmoid function (formula (8)). The inertia weight $w$ decreases linearly from 0.9 to 0.4 during the whole process. The learning factors $c_1$ and $c_2$ are set to 2.8 and 1.3 respectively. The maximum iteration number $T_{\max}$ is set to 1000.

$$f(u)=\frac{2}{1+\exp(-2\times u)}-1, \quad -\infty < u < +\infty, \tag{7}$$

$$f(u)=\frac{1}{1+\exp(-u)}, \quad -\infty < u < +\infty. \tag{8}$$

Speedup ratio $S$ is the most commonly-used index to measure the acceleration performance. $S$ is defined as the ratio of $T_{\mathrm{CPU}}$ (the running time of the CPU-based program) and $T_{\mathrm{GPU}}$ (the running time of the GPU-based program) under the condition of the same number of particles and the same number of iterations in the PSO-ANN algorithm:

$$S=\frac{T_{\mathrm{CPU}}}{T_{\mathrm{GPU}}}. \tag{9}$$

To get the running time, clock() function is used on CPU side and cudaEventElapsedTime() function based on "Events" is used on GPU side. Considering the influence caused by randomness, the program is run 20 times repeatedly under the circumstance of the same number of particles whenever on CPU or GPU side, and the result is their average value. To ensure the computing precision, all decimals use double precision on both CPU side and GPU side.

The experimental results are shown in Table 5 and Column 7-17 of Table 2. It's worth noting that the meanings of the "sum of the absolute error" (in Table 3 and Table 4) and the "average sum of the absolute error" (in Table 5) are different. The "sum of the absolute error" means the sum of the absolute error of the average of the experimental values and the theoretical values. The "average sum of the absolute error" means the average of the sum of the absolute error of the experimental values and the theoretical values. We believe that compared to the "sum of the absolute error", the "average sum of the absolute error" can be easily got by calculation, objectively reflects the results in each experiment, and is greater than the "sum of the absolute error" under the same conditions. In other words, if the "average sum of the absolute error" in Table 5 is superior to the "sum of the absolute error" in Table 4, the "average sum of the absolute error" in Table 5 is certainly superior to the "average sum of the absolute error" corresponding in Table 4. In Column 7-17 of Table 2, each column gives particular experimental values from GPU-based algorithm, the "sum of the absolute error" of which is closest to the "average sum of the absolute error" in Table 5.

Table 5: Speedup ratio achieved by parallel PSO-ANN when modeling the resonant frequency of rectangular MSA

| Number of Particles | Running Time/s | | Average Sum of the Absolute Error/MHz | | Speedup Ratio |
|---|---|---|---|---|---|
| | CPU | GPU | CPU | GPU | |
| 32 | 1.309 | 1.680 | 3171.6 | 3259.4 | 0.8 |
| 64 | 2.495 | 1.672 | 2559.1 | 2534.0 | 1.5 |
| 128 | 4.839 | 1.672 | 2189.3 | 2200.4 | 2.9 |
| 256 | 9.659 | 1.680 | 1694.5 | 1697.4 | 5.7 |
| 512 | 19.141 | 1.686 | 1331.9 | 1352.5 | 11.4 |
| 1024 | 38.267 | 1.717 | 1215.0 | 1267.6 | 22.3 |
| 2048 | 77.445 | 1.920 | 1098.7 | 1103.3 | 40.3 |
| 4096 | 155.406 | 2.186 | 974.2 | 997.5 | 71.1 |
| 8192 | 309.941 | 2.636 | 863.9 | 895.2 | 117.6 |
| 16384 | 620.352 | 2.824 | 795.1 | 885.7 | 219.7 |
| 32768 | 1241.120 | 5.647 | 722.8 | 840.1 | 219.8 |
| 65536 | 2481.003 | 8.768 | 689.4 | 792.9 | 283.0 |
| 131072 | 4958.867 | 15.160 | 674.7 | 765.0 | 327.1 |

We make some analysis on Table 3, Table 4 and Table 5:

(a) Generally, ANN models have obvious advantages over traditional methods in calculation precision. GPU-based parallel PSO-ANN has obvious advantages over CPU-based sequential PSO-ANN in running speed.

(b) The more number of particles, the higher speedup ratio. Compared with CPU-based sequential PSO-ANN, 327 times of maximum speedup ratio has achieved in GPU-based parallel PSO-ANN. When the number of particles doubles, the speedup ratio roughly doubles if the number of particles is less than 16384 (The maximum number of resident threads on this GPU is 26624.), and increases at a relatively slow rate if the number of particles is more than 32768.

(c) Compared with CPU-based sequential PSO-ANN, GPU-based parallel PSO-ANN has the same optimization stability. When the number of particles increases, the error of CPU-based program and the error of GPU-based program both decreases. The error of CPU-based program and the error of GPU-based program are similar or same under the condition of the same number of particles.

(d) Substantially increasing the number of particles on GPU side is a special method, which adapts to the CUDA programming model. The runtime increases very limitedly when substantially increasing the number of particles on GPU side. The error of GPU-side parallel PSO-ANN is superior to the results of [14,15] when the number of particles is greater than or equal to 128, superior to the results of [19] when the number of particles is greater than or equal to 256, superior to the results of all the traditional methods including [13] when the number of particles is greater than or equal to 512, superior to the results of [18] when the number of particles is greater than or equal to 4096, and superior to the results of all the literatures including [20] when the number of particles is greater than or equal to 32768.

We provide the following suggestions as reference for GPU-based parallel PSO-ANN algorithm:

(1) For the standard PSO-ANN algorithm in this study, the network error can be significantly reduced with the very limited runtime increment when substantially increasing the number of particles on GPU side.

(2) Other types of improved PSO-ANN algorithm do not always adapt to the GPU parallel architecture. The algorithm performance can be further improved if the improved PSO-ANN algorithm is suitable to parallelize on GPU side.

## VI. CONCLUSION

The CUDA-based parallel PSO-ANN scheme is designed to rapidly model the resonant frequency of rectangular MSA. The proposed approach corresponds one particle to one thread, and deals with a large number of GPU threads in parallel to greatly save computing time and improve computing accuracy. The experiments show that the modeling runtime can be greatly reduced when parallelizing the PSO-ANN algorithm on GPU side. Furthermore, the network error can be significantly reduced with the very limited runtime increment when substantially enlarging the number of particles on GPU side. The proposed GPU-based parallel PSO-ANN in this paper can be extended to other similar microwave engineering designs easily.

## ACKNOWLEDGMENT

This work is supported partly by Pre-research foundation of the shipping industry of China under grant No. 10J3.5.2 and Project funded by the priority academic program development of Jiangsu higher education institutions.

## REFERENCES

[1] K. L. Wong, "Compact and broadband microstrip antennas," *New York: John Wiley & Sons Inc.*, 2002.

[2] D. G. Fang, "Antenna theory and microstrip antennas," *Beijing: Science Press*, 2007 (in English).

[3] J. Q. Howell, "Microstrip antennas," *IEEE Transactions on Antennas and Propagation*, vol. 23, no. 1, pp. 90-93, 1975.

[4] E. O. Hammerstad, "Equations for microstrip circuits design," *Proc. 5th Eur. Microw. Conf.*, Hamburg, Germany, pp. 268-272, September 1975.

[5] K. R. Carver, "Practical analytical techniques for the microstrip antenna," *Proc. Workshop Printed Circuit Antenna Tech.*, New Mexico State Univ., Las Cruces, NM, 7.1-7.20, October 1979.

[6] I. J. Bahl and P. Bhartia, "Microstrip antennas," *MA: Artech House*, 1980.

[7] J. R. James, P. S. Hall, and C. Wood, "Microstripa antennas-theory and design," *London: Peregrinus*, 1981.

[8] D. L. Sengupta, "Approximate expression for the resonant frequency of a rectangular patch antenna," *Electronics Letters*, vol. 19, no. 20, pp. 834-835, 1983.

[9] R. Garg and S. A. Long, "Resonant frequency of electrically thick rectangular microstrip antennas," *Electronics Letters*, vol. 23, no. 21, pp. 1149-1151, 1987.

[10] W. C. Chew and Q. Liu, "Resonance frequency of a rectangular microstrip patch," *IEEE Transactions on Antennas Propagation*, vol. 36, no. 8, pp. 1045-1056, 1988.

[11] K. Guney, "A new edge extension expression for the resonant frequency of electrically thick rectangular microstrip antennas," *Int. J. Electron.*, vol. 75, pp. 767-770, 1988.

[12] M. Kara, "Closed-form expressions for the resonant frequency of rectangular microstrip antenna elements with thick substrates," *Microwave and Optical Technology Letters*, vol. 12, no. 3, pp. 131-136, 1996.

[13] K. Guney, "A new edge extension expression for the resonant frequency of rectangular microstrip antennas with thin and thick substrates," *J. Commun. Tech. Electron.*, vol. 49, no. 1, pp. 49-53, 2004.

[14] K. Guney, S. Sagiroglu, and M. Erler, "Generalized neural method to determine resonant frequencies of various microstrip antennas," *International Journal of RF and Microwave Computer-Aided Engineering*, vol. 12, no. 1, pp. 131-139, 2002.

[15] S. Sagiroglu and A. Kalinli, "Determining resonant frequencies of various microstrip antennas within a single neural model trained using parallel tabu search algorithm," *Electromagnetics*, vol. 25, no. 6, pp. 551-556, 2005.

[16] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 69-73, 1998.

[17] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization: an overview," *Swarm Intelligence*, vol. 1, no. 1, pp. 33-57, 2007.

[18] Y. B. Tian, Z. Q. Li, and J. H. Wang, "Model resonant frequency of rectangular microstrip antenna based on particle swarm neural network," *Journal of Microwaves*, vol. 25, no. 5, pp. 45-50, 2009 (in Chinese).

[19] Y. Dong and Y. B. Tian, "Modeling resonant frequency of microstrip antenna based on neural network trained by PSO-BP algorithm," *Journal of Communication University of China (Science and Technology)*, vol. 16, no. 2, pp. 58-63, 2009 (in Chinese).

[20] Y. B. Tian and Y. Dong, "Modeling resonant frequency of microstrip antenna based on neural network ensemble," *Chinese Journal of Radio Science*, vol. 24, no. 4, pp. 610-616, 2009 (in Chinese).

[21] M. Pethick, et al., "Parallelization of a backpropagation neural network on a cluster computer," *The Fifteenth IASTED International Conference on Parallel and Distributed Computing and Systems*, pp. 574-582, 2003.

[22] G. Singhal, A. Jain, and A. Patnaik, "Parallelization of particle swarm optimization using message passing interfaces (MPIs)," *IEEE*

*World Congress on Nature & Biologically Inspired Computing*, pp. 67-71, 2009.

[23] K. Deep, S. Sharma, and M. Pant, "Modified parallel particle swarm optimization for global optimization using message passing interface," *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications*, pp. 1451-1458, 2010.

[24] D. Z. Wang, et al., "Parallel multi-population particle swarm optimization algorithm for the uncapacitated facility location problem using openMP," *IEEE Congress on Evolutionary Computation*, pp. 1214-1218, 2008.

[25] G. K. Venayagamoorthy and V. G. Gudise, "Swarm intelligence for digital circuits implementation on field programmable gate arrays platforms," *IEEE Conference on Evolvable Hardware*, pp. 83-86, 2004.

[26] Y. Maeda and N. Matsushita, "Simultaneous perturbation particle swarm optimization using FPGA," *IEEE International Joint Conference on Neural Networks*, pp. 2695-2700, 2007.

[27] L. Veronese and R. Krohling, "Swarm's flight: accelerating the particles using C-CUDA," *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 3264-3270, 2009.

[28] P. Kromer, J. Platos, and V. Snasel, "A brief survey of advances in particle swarm optimization on graphic processing units," *2013 IEEE World Congress on Nature and Biologically Inspired Computing*, pp. 182-188, 2013.

[29] K. Ganeshamoorthy and D. N. Ranasinghe, "On the performance of parallel neural network implementations on distributed memory architectures," *8th IEEE International Symposium on Cluster Computing and the Grid*, pp. 90-97, 2008.

[30] M. Kara, "The resonant frequency of rectangular microstrip antenna elements with various substrate thicknesses," *Microwave and Optical Technology Letters*, vol. 11, no. 2, pp. 55-59, 1996.

**Feng Chen** (1989-) graduate student. His main research is high performance computing and computational intelligence technologies and its applications in electronics and electromagnetism. Presently at the School of Electronics and Information, Jiangsu University of Science and Technology, Zhenjiang 212003, P. R. China.

**Yu-bo Tian** (1971-), received his Ph. D. in 2004 from Nanjing University. Now, he is a Full Professor at Jiangsu University of Science and Technology. His main research concerns computational intelligence and swarm intelligence technologies and applications in electronics and electromagnetism.