Parallel Computation Based Neural Network Approach for Parametric Modeling of Microwave Circuits and Devices

by

Shunlu Zhang, B. Eng

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Carleton University
Ottawa, Ontario

Your file Votre référence
ISBN: 978-0-494-93638-2

Our file Notre référence
ISBN: 978-0-494-93638-2

Abstract

This thesis presents a wide-range parametric modeling technique utilizing enhanced Parallel Automatic Data Generation (PADG) and Parallel Multiple ANN Training (PMAT) techniques. A Parallel Model Decomposition (PMD) technique is proposed for neural network models with wide input parameter ranges. In this technique, wide ranges of input parameters are decomposed into small sub-ranges. Multiple neural networks with simple structures, hereby referred as sub-models, are trained to learn the input-output relationship with their corresponding sub-ranges of input parameters. A frequency selection method has been proposed to reduce the sub-models training time and increase the accuracy of sub-models. Once developed, these sub-models cover the entire ranges of parameters and provide an accurate model for microwave components with wide ranges of parameters. A Quasi-Elliptic filter example is used to illustrate the validity of this technique.

The PADG technique and PMAT technique exploit the full utilization of a parallel computational platform that consists of multiple computers with multiple processors. Task distribution strategies have been proposed for both techniques. The proposed techniques have achieved remarkable speed gains against the conventional neural network data generation and training processes. A parallel Back Propagation training implementation using multiple graphics processing units is proposed for the first time. A modular neural network application example has been presented to demonstrate the advantages of PMAT techniques.

Acknowledgements

I would like to express my sincere thanks to my thesis supervisor Professor Qi-Jun Zhang and co-supervisor Professor Pavan Gunupudi for their professional guidance, invaluable inspiration, motivation, suggestion and patience throughout the research work and preparation of this thesis. I am highly indebted to them for having trained me into a full-time researcher with technical, computation and presentation skills, and professionalism. Their leadership and vision for quality research and developmental activities has made the pursuit of this thesis a challenging, enjoyable and stimulating experience.

My deep appreciation is given to Yazi Cao for his enthusiasm, promotional skills and helpful discussion. I wish to thank my colleagues Venu-Madhav-Reddy Gongal-Reddy and Sayed Alireza Sadrossadat for reading the manuscript and for many helpful suggestions to improve the thesis.

Many thanks to Blazenka Power, Anna Lee, Sylvie Beekmans, Scott Bruce, Nagui Mikhail, Khi Chivand all other staff and faculty for providing the excellent lab facilities and friendly environment for study and research.

This thesis would not have been possible without years of support and encouragement from my parents. This thesis is dedicated to them for their endless love. Last but not least, I would like to thank my girlfriend and my friends. Their love, support and encouragement is the source of strength for overcoming any difficulty and achieving success in my whole life.

Table of Contents

Abstract i

Acknowledgements ii

Table of Contents iii

List of Figures vii

List of Tables ix

List of Acronyms xi

Chapter 1: Introduction

1.1 Motivations 1

1.2 Thesis Contributions 4

1.3 Organization of the Thesis 5

Chapter 2: Literature Review and Background

2.1 Neural Network Applications in RF/Microwave Design 7

2.2 Neural Network Model Development Overview 8

2.2.1 Neural Network Structures 9

2.2.2 Neural Network Data Generation 13

2.2.3 Neural Network Training 14

2.2.3.1 Training Objective 14

2.2.3.2 Back Propagation Training Algorithm 15

2.2.3.3 Conjugate Gradient Training Algorithm 17

2.2.3.4 Quasi-Newton Training Algorithm 18

2.3 Overview of Parallel Computing Architectures 19

2.3.1 Hybrid Distributed-Shared Memory Architecture 20

2.3.2 Hybrid Shared Memory-Graphics Processing Units Architecture 24

Chapter 3: Parallel Automatic Data Generation

3.1 Introduction 27

3.2 Key Aspects of Parallel Automatic Data Generation 28

3.3 Task Distribution Strategy 31

3.4 Verification of Parallel Automatic Data Generation 34

3.5 Summary 43

Chapter 4: Parallel Multiple ANN Training

4.1 Introduction 44

4.2 Parallel Multiple ANNs Training on Central Processing Unit 45

4.3 Parallel Multiple ANNs Training on Graphics Processing Unit 49

4.4 Verification of Parallel Multiple ANN Training 62

4.5 Summary 65

Chapter 5: Wide-Range Parametric Modeling Technique for Microwave Components Using Parallel Computational Approach

5.1 Introduction 66

5.2 Proposed Parallel Model Decomposition Technique 67

5.3 Application Example of a Quasi-Elliptic Filter 71

5.3.1 50 ~ 70GHz with Global Frequency Range for Each Sub Model 71

5.3.2 40 ~ 80GHz with Local Frequency Range for Each Sub Model 77

5.4 Summary 84

Chapter 6: Conclusions and Future Research

6.1 Conclusions 86

6.2 Future Research 88

Bibliography 90

List of Figures

Figure 2.1 Structure of a three layer feedforward Multilayer Perceptrons (MLP) 11

Figure 2.2 Structure of Hybrid Distributed-Shared Memory Architecture 21

Figure 2.3 Hybrid Shared Memory-Graphics Processing Units Architecture 25

Figure 3.1 Framework of Parallel Automatic Data Generation (PADG) on hybrid distributed-shared memory system 29

Figure 3.2 Structure of interdigital band-pass filter with four design variables 34

Figure 4.1 Framework of Parallel Multiple ANNs Training on Central Processing Unit (PMAT-C) on Hybrid Distributed-Shared Memory Architecture (HDSMA) 46

Figure 4.2 Speed Up for Matrix Multiplication by CUDA and Intel MKL 51

Figure 4.3 CUDA operations time for matrix multiplication with matrix size of N 53

Figure 4.4 Framework of Parallel Multiple ANN Training on multiple GPUs 61

Figure 4.5 Structure of a cavity microwave bandpass filter with structural decomposition 63

Figure 5.1 Framework of proposed Parallel Model Decomposition (PMD) technique ...68

Figure 5.2 Structure of a Quasi-Elliptic filter. This filter model has four wide range geometrical parameters as inputs. 71

Figure 5.3 Comparison of outputs of the proposed ANN model and EM simulation for four filters with parameters belong to four different sub-models. 76

Figure 5.4 Magnitude of S11 of 32 training data sets before applying frequency range refinement. The 32 training data sets are generated for one sub-model 81

Figure 5.5 Magnitude of S11 of 32 training data sets after applying frequency range refinement. The 32 training data sets are generated for one sub-model 82

Figure 5.6 Comparison of outputs of the proposed ANN model and EM simulation for six filters with parameters belong to six different sub-models. 85

List of Tables

Table 3.1 Geometric value of design parameters of interdigital band-pass filter 38

Table 3.2 Speed up and efficiency of executing Parallel Automatic Data Generation on Shared Memory Architecture 39

Table 3.3 Speed up and efficiency of executing Parallel Automatic Data Generation on Distributed Memory Architecture 40

Table 3.4 Speed up and efficiency of executing Parallel Automatic Data Generation on Hybrid Distributed-Shared Memory Architecture 42

Table 4.1 Routines of Parallel Multiple ANNs Training on Graphics Processing Units (PMAT-G) 59

Table 4.2 Speed up of executing Parallel Multiple ANN Training 64

Table 5.1 Comparison of training results for 24 sub-models with 50 ~ 70 GHz global frequency range 72

Table 5.2 Comparison of data generation time for 24 sub-models with 50 ~ 70 GHz global frequency range 74

Table 5.3 Comparison of ANN training time for 24 sub-models with 50 ~ 70 GHz global frequency range 74

Table 5.4 Comparison of ANN training data generation time for 108 sub-models 78

Table 5.5 Comparison of ANN training results for 108 sub-models with global frequency range of 40 ~ 80 GHz 78

Table 5.6 Comparison of proposed PMD ANN training results and training time by parallel multiple ANN training on CPU before and after frequency selection 83

Table 5.7 Comparison of ANN training time for 108 sub-models with local frequency ranges 83

List of Acronyms

ADG:	Automatic Data Generator
ANN:	Artificial Neural Networks
API:	Application Programming Interface
BP:	Back Propagation
CAD:	Computer Aided Design
CPU:	Central Processing Unit
CUDA:	Compute Unified Device Architecture
DMA:	Distributed Memory Architecture
DOE:	Design of Experiments
EM:	Electromagnetic
GPGPU:	General-Purpose computing on Graphics Processing Units
GPU:	Graphics Processing Units
HDSMA:	Hybrid Distributed-Shared Memory Architecture
HSMGPUA:	Hybrid Shared Memory-Graphics Processing Units Architecture

Intel MKL:	Intel Math Kernel Library
KBNN :	Knowledge Based Neural Networks
MLP:	Multilayer Perceptrons
MNN:	Modular Neural Networks
MPI:	Message Passing Interface
OpenCL:	Open Computing Language
OpenMP:	Open Multiprocessing
PADG:	Parallel Automatic Data Generator
PAMG:	Parallel Automatic Model Generation
PMAT-C:	Parallel Multiple ANNs Training on Central Processing Unit
PMAT-G:	Parallel Multiple ANNs Training on Graphics Processing Units
PMD:	Parallel Model Decomposition
RBF:	Radial Basis Function
RF:	Radio Frequency
SMA:	Shared Memory Architecture
SOM:	Self-organizing Maps

Chapter 1

Introduction

1.1 Motivations

With the efficient use of Computer Aided Design (CAD) tools, the design of Radio Frequency (RF) and microwave circuits and systems has seen a considerable growth. Behaviors of circuits and systems, such as performance, stability, reliability and manufacturability, etc., can be predicted by CAD tools before hardware implementation. However, as the signal frequency increases, the microwave/RF design becomes more complicated. Even simple devices and circuits at high frequency may require relatively complex models to correctly predict their high frequency behaviors. Modeling these complex models is considered to be a major challenge in the CAD implementation. The conventional electrical models are no longer accurate. The electromagnetic (EM) effect becomes the necessary element to be included in the accurate models. However, detailed EM simulations have been well recognized computationally intensive. There is growing need to develop fast and reliable CAD tools to meet the advanced requirements in RF and microwave design areas.

In recent years, Artificial Neural Networks (ANNs) have been recognized as a powerful tool for RF and microwave modeling and design [1]. ANNs have been applied to a wide variety of applications in RF and microwave area, such as passive microwave structures [2], transistors [3], antennas [4], amplifiers [5], waveguide filters [6], microwave optimization [7], etc. ANNs can be developed and trained from physics/EM simulation or measurement data by learning input-output relationships. The trained models can be used to represent previously learned physics/EM behaviors, and can also be used to respond to the new data that has not been used for ANN development. Due to the particular learning ability, ANNs can generalize physics/EM behaviors and provide fast and accurate solution to the new data. ANNs can be more accurate than the polynomial regression models [8], handle more dimensions than the look-up tables [9], faster than detailed EM simulation [10], and easier to develop when a new device/technology is introduced [11].

With continuing development in the applications of ANN to microwave design, there is growing need to reduce the cost of ANN model development. During the model development using neural network, more data points in the model input parameter space are required to best represent the target problems when they become complicated. In other words, the amount of the training data is determined by the complexity of the target problem. As a result, data generation can be expensive because of the need for a large amount of training data. Many advanced techniques has been proposed to reduce the amount of training data for developing neural network models while keeping a high level of model accuracy, such as knowledge based neural networks (KBNN) [12], space-mapping neural networks [13] [14]. Another direction is to develop an automatic tool to

accelerate data generation. Parallel computational techniques have been applied to reduce the cost of training data generation by driving multiple EM/physics/circuit simulators simultaneously on multiple processors on one computer [15]. With the development of computer technology, computers can be interconnected by local networks to share the work load [16]. Parallel computational techniques for interconnected computers can also be applied to training data generation to further reduce the cost of EM/physics/circuit simulations. One motivation of the thesis is to take full use of interconnected computer resources to accelerate training data generation.

With the increase in the complexity of RF/microwave components structures, the dimensions of the inputs and the outputs of neural network are increasing. Modular neural networks (MNN) have been developed to address the challenge of high-dimensional modeling problems [17] [18]. This technique decomposes a complex RF/microwave component structure into several simple substructures then develops several simple sub-neural-network modules, hereby referred as sub-models. These sub-models are combined with equivalent-circuit model to produce an approximate solution of the entire component. The main objective of modular neural network is to develop a high-dimensional neural network model, which is too expensive to develop using a conventional neural network approach. The conventional method to train modular neural networks is to train one sub-model after another. In order to reduce the cost of sub-models training, applying parallel computational technique to the modular neural network training is another motivation of the thesis.

As the signal frequency increases, the RF/microwave design becomes more and more complex. Design parameters need to be searched in a wide range to address the

design optimization challenge over a wide frequency range. Existing neural network techniques often become inefficient for microwave components with wide-range parameters. Developing an efficient and accurate parametric modeling technique for neural networks with wide input parameter ranges is the third motivation of the thesis.

1.2 Thesis Contributions

As stated in the thesis's motivation, the main contribution of this thesis is to develop an efficient and accurate parametric modeling technique towards wide input parameter problems utilizing parallel data generation and parallel multiple ANN training techniques. In this thesis, the following works are presented:

(1) The development of an enhanced data generator making use of interconnected computer resources. A unified task distribution algorithm based on the available computational resources is presented. The parallel automatic data generation reduces human labor and achieves high speed gains for the neural network data generation stage. The advantages of utilizing this technique are demonstrated through the comparison by the proposed technique and the existing parallel data generation technique.

(2) Two parallel computational approaches for multiple neural network training are explored to accelerate the sub-models training process of modular neural networks. We propose to train multiple sub-models simultaneously on a group of interconnected computers. Another novel ANN training approach, called parallel multiple ANN training on Graphics Processing Units (GPUs), is pioneered by

parallelizing Back Propagation (BP) training algorithm on multiple GPUs. Example of utilizing both techniques for modular neural network training of a microwave cavity filter is demonstrated.

(3) Wide-range parametric modeling technique using parallel computational approaches is proposed. This technique decomposes the wide input parameter ranges of neural network into smaller parts and develops multiple independent sub-models with simple structures. Training data generation and sub-models training are executed concurrently on a group of interconnected computers. A frequency selection strategy is proposed to determine the working frequency range of each sub-model. Each sub-model is only trained with training data samples inside its working frequency range. The accuracy and efficiency of sub-models are further increased after applying frequency selection strategy. This technique provides an efficient and accurate solution to address the challenge of developing neural network with wide input parameter ranges.

1.3 Organization of the Thesis

The thesis is organized as follows.

In chapter 2, the procedures of neural network model development are reviewed at the beginning. Neural network structure and the training algorithm, which are the two major issues in developing neural network models, are described in detail. Two kinds of hybrid parallel computational architectures and programming interfaces are briefly

reviewed. Existing techniques to accelerate neural network data generation and training are explained.

In chapter 3, an enhanced data generation technique is proposed. The new efficient data generate technique is aimed at generation of massive data for neural network without intensively using human labor while achieving high performance gains.

In chapter 4, parallel multiple ANN training techniques on CPU and GPU are explored. These techniques aim at accelerating sub-models training stage of modular neural network. The speed up is proven by comparing the sub-models training time of the two proposed techniques with the conventional ANN training method.

Chapter 5 introduces a novel neural network model decomposition technique. This technique addresses the challenge of developing an efficient and accurate neural network model with wide input parameter ranges. A microwave filter example is provided to demonstrate the efficiency and validity of the proposed technique.

Finally, conclusions of the thesis are presented in chapter 6. Recommendations for applying parallel techniques towards other neural network algorithms and developing an intelligent model decomposition technique are also made.

Chapter 2

Literature Review and Background

2.1 Neural Network Applications in RF/Microwave Design

The rapid development in the RF/microwave industry has led to needs of creating efficient statistical design techniques, which places enhanced demands on Computer-Aided Design (CAD) tools for RF/microwave designs [19]. During the CAD process, the most critical step is to develop efficient and accurate models of RF/microwave circuits and components [20]. A variety of modeling approaches have been introduced for RF/microwave components [21][22][23]. Some of them are computationally efficient but are short of accuracy or are limited in the degree of nonlinearity. Hence, they are not considered to be suitable models for RF/microwave design. Detailed electromagnetic (EM) and physics models of active/passive components offer excellent accuracy, but the models are computationally intensive, which limits their application in RF/microwave design.

Recently, Artificial Neural Networks (ANNs) technology has been introduced as an unconventional technology in the RF/microwave modeling, simulation and optimization. ANN processes the ability of learning from samples of input-output data and establishes

accurate nonlinear relationships [1]. ANN has been proved to have distinguished advantage of being both fast and accurate. In the past few years, ANN has been widely used in a variety of RF/microwave design, such as microstrip interconnects [12], vias [24], spiral inductors [25], FET devices [26], HBT devices [27], HEMT devices [28], coplanar waveguide (CPW) circuit components [2], mixers [29], embedded components [30][31][32], packaging and interconnects [33], etc. Neural network has also been used in circuit simulation and optimization [10][34], signal integrity analysis and optimization of high-speed VLSI interconnects [33][35], microstrip circuit design [36], process design [37], circuit synthesis [38], EM-optimization [39], global modeling [40] and microwave impedance matching [41]. These pioneering works have established the framework of neural network modeling technique for both device level and circuit level in RF/microwave design.

2.2 Neural Network Model Development Overview

There are four major stages involved in the neural network model development: problem identification, data generation, model training and model testing.

The first stage is the identification of the model inputs and outputting based on the purpose of the model. A certain neural network structure should be properly selected to ensure the efficient development of neural network model.

The second stage of neural network model development is data generation. The data range should be first defined depending on the target problems. Data generation is executed by RF/microwave simulators or measurement to obtain the outputs for each input sample. The number of input samples to be generated should be carefully decided so that the developed neural network model can best represent the target problem.

After the training data generation stage is finished, the next stage is ANN training. Neural network learns the input-output relationship by iteratively update the weighting in the ANN training algorithm. Once trained, the ANN model can be used as an efficient and accurate model.

ANN testing is implemented in the final stage. Since testing data has not been used by the neural network model, ANN testing is used to determine the performance of neural networks prediction of the outputs for the new input data.

2.2.1 Neural Network Structures

The first step in the neural network development is to identify a suitable neural network structure. A neural network has two types of components: processing elements called neurons and connections between neurons known as links. It is important to determine the size of neural network structure, i.e., the number of neurons and the number of hidden layers, to deal with the problems of under-learning and over-learning.

Under-learning refers to a small size neural network that cannot learn the target problem very well, which is usually caused by insufficient hidden neurons. Over-learning refers to a large size neural network that can match the training data very well but cannot generalize well to match with the validation data. The reason is that too many hidden neurons with insufficient training data lead to too much freedom in the input-output relationship represented by a neural network. Hence, many trails of error may be required to select a proper neural network structure to meet the desired accuracy of the neural network model.

A variety of neural network structures have been developed for RF/microwave design, such as multilayer perceptrons (MLP), radial basis function (RBF) neural network, wavelet neural network, self-organizing maps (SOM) and recurrent neural networks. Feedforward neural network is a basic type of neural network, which is capable of approximating generic and integrable functions [1]. The most popular type of neural network structure is MLP, which is a feedforward structure that has three typical types of layers: an input layer, one or more hidden layers and an output layer as shown in Figure 2.1. Suppose the total number of layers is L. The 1st layer is input layer, the 2nd to $(L-1)^{th}$ layers are the hidden layers and the L^{th} layer is the output layer. Let the number of neurons in the l^{th} layer be N_l, $l = 1, 2, \ldots, L$. Let w_{ij}^{l} represent the weight parameter of the link between the j^{th} neuron of the $(l-1)^{th}$ layer and i^{th} neuron of the l^{th} layer. Let w_{i0}^{l} represent the value of i^{th} neuron in the l^{th} layer when all the previous hidden layer neuron

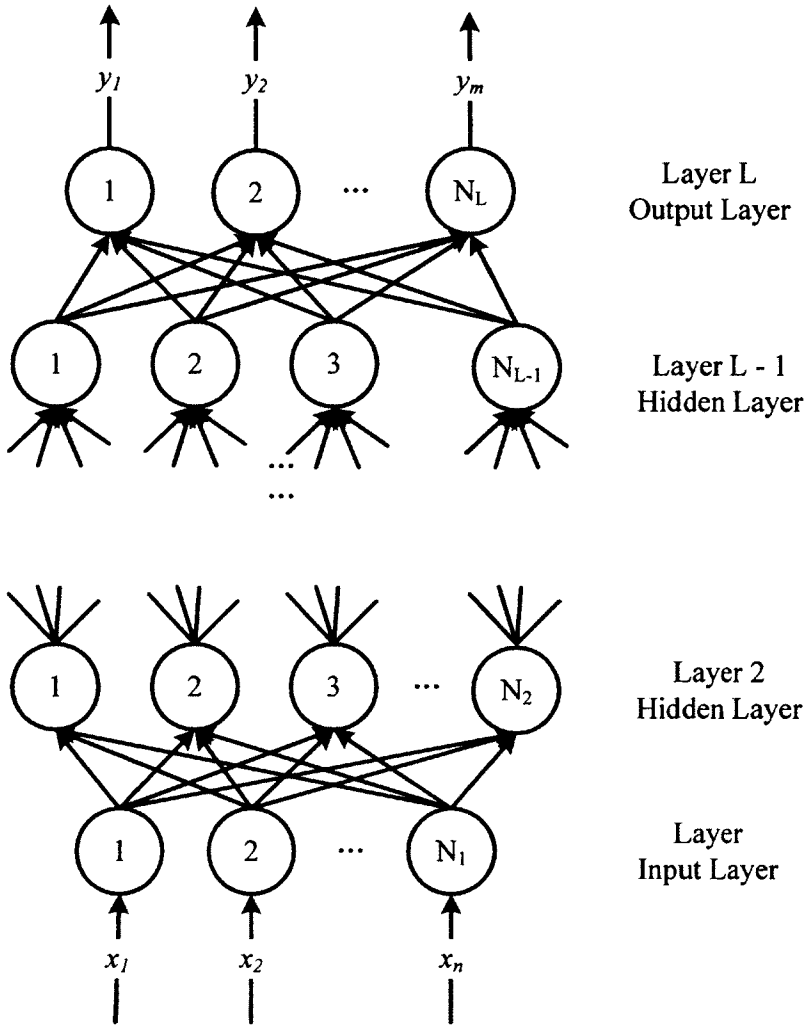


Figure 2.1 Illustration of the feedforward Multilayer Perceptrons (MLP) structure. The MLP structure typically consists of one input layer, one or more hidden layers and one output layer.

responses are zero, which is known as bias.

Given the inputs $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^T$. Let x_i represent the i^{th} input parameter. Let z_i^l represent the i^{th} neuron of l^{th} layer, which can be computed according to the standard MLP formulae as

$$z_i^1 = x_i, i = 1, 2, \ldots, N_1, N_1 = n \tag{2.1}$$

$$z_i^l = \sigma\left(\sum_{j=0}^{N_{l-1}} w_{ij}^l z_j^{l-1}\right), i = 1, 2, \ldots, N_l, l = 2, 3, \ldots, L \tag{2.2}$$

where $\sigma(\bullet)$ is the activation function of hidden neurons. The most commonly used activation function is the logistic sigmoid function given by

$$\sigma(\gamma) = \frac{1}{(1+e^{-\gamma})} \tag{2.3}$$

which has the property

$$\sigma(\gamma) \to \begin{cases} 1 & \gamma \to +\infty \\ 0 & \gamma \to -\infty \end{cases} \tag{2.4}$$

Other possible activation functions of $\sigma(\bullet)$ are the arctangent function

$$\sigma(\gamma) = \left(\frac{2}{\pi}\right) arctan(\gamma) \tag{2.5}$$

and the hyperbolic tangent function

$$\sigma(\gamma) = \frac{e^{\gamma} - e^{-\gamma}}{e^{\gamma} + e^{-\gamma}} \tag{2.6}$$

The neural network outputs are represented by $y = [y_1, y_2, \ldots, y_m]^T$. The value of i^{th} neuron in the output layer can be obtained as

$$y_i = z_i^L, i = 1, 2, \ldots, N_L, N_L = m \tag{2.7}$$

where linear activation function is implied for output neurons, which is most suitable for RF and microwave modeling problems.

2.2.2 Neural Network Data Generation

Data generation plays an important role in the development of neural network models. Neural network models are considered as black box models. Certain relationship between the inputs and outputs are established by the internal learning feature of neural networks. The inputs of neural network models are geometrical/physical parameters, such as length, width, frequency, etc. The outputs are obtained design results, such as real and imaginary parts of S-parameters, etc. The input data ranges should be first defined depending on the target problems. The training data generation is to obtain a set of data to provide for neural network models to learn the input-output relationship, which are sampled slightly beyond the determined input data ranges. During neural network training, a set of validation data is required to monitor the training quality and give indication to terminate neural network training. After neural network training is completed, a set of testing data, with input values within the defined input data ranges, is

used to check the final quality of neural network models.

Training data, validation data and testing data can either be measured data or simulated data. In order to ensure the accuracy of neural network models, sufficient data should be measured or simulated. In the microwave/RF design, data are usually obtained by detailed software simulators. With the increased complexity in the structures of microwave devices and circuits, the time consumption on simulation is keep increasing. Various techniques have been introduced to reduce the quantity of data required for developing an accurate neural network model [12][13][14]. Parallel computational approaches have also been introduced for data generation to accelerate the software simulators [42]. With the development of computer technology, more speed gains against conventional method are expected to achieve by applying the latest computer technology to neural network data generation stage.

2.2.3 Neural Network Training

2.2.3.1 Training Objective

A neural network model can be developed through an optimization process known as training. When all the input information is feedforwarded to the output layer, the neural can start to be trained. Let $\boldsymbol{d}_k$ be a vector representing the desired output of the k^{th} training sample. Let $\boldsymbol{w} = [\, w_{10}^2,\ w_{11}^2,\ w_{12}^2,\ \ldots,\ w_{N_L N_{L-1}}^L \,]^T$ representing all weight parameters in the

neural network model. The training objective is to minimize the difference between neural network outputs and desired outputs, known as error, through updating the weights. In other words, the neural network training target is to find an optimal set of weights such that the error is minimized as

$$\min_{\boldsymbol{w}} E(\boldsymbol{w}) = \frac{1}{2}\sum_{k\in T_r}\sum_{j=1}^{m}(y_j(\boldsymbol{x}_k,\boldsymbol{w}) - d_{jk})^2 \tag{2.8}$$

where T_r is an index set of training data, d_{jk} is the j^{th} element of $\boldsymbol{d}_k$, $y_j(\boldsymbol{x}_k, \boldsymbol{w})$ is the j^{th} neural network output for the inputs of the k^{th} training sample.

There are various kinds of training algorithms. Each algorithm has its own scheme for updating the weights. Three typical training algorithms: Back Propagation, Conjugate Gradient and Quasi-Newton are reviewed as follows.

2.2.3.2 Back Propagation Training Algorithms

The Back Propagation (BP) is the most popular algorithm for neural network training [43]. The BP algorithm calculates the derivation of the cost function $E(\boldsymbol{w})$ to weights $\boldsymbol{w}$ layer by layer. The weights of neural network $\boldsymbol{w}$ can be updated along the negative direction of the gradient of $E(\boldsymbol{w})$ as

$$\Delta \boldsymbol{w}_{now} = -\eta \left.\frac{\partial E_k(\boldsymbol{w})}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}_{now}} + \alpha(\boldsymbol{w}_{now} - \boldsymbol{w}_{old}) \tag{2.9}$$

$$\Delta w_{now} = -\eta \left. \frac{\partial E_{T_r}(w)}{\partial w} \right|_{w=w_{now}} + \alpha(w_{now} - w_{old}) \qquad (2.10)$$

where E_k is the error of the k^{th} training sample, E_{Tr} is the error of all training samples, subscription *now*, *old* are the current and previous values of weights, η is the learning rate and α is the momentum factor added to avoid the oscillation of the weighting during the training process [44]. In Equation (2.9), the weights are updated after each training sample is applied to the neural network, which is called update sample-by-sample. In Equation (2.10), the weights are updated after all training samples are applied to the neural network, which is known as batch mode update.

Various techniques have been introduced to accelerate the Back Propagation training process. One aspect is to dynamically adapt the learning rate η, which controls the step size of weight update, based on the number of training epoch [45][46] or based on the training errors [47]. Another aspect is to implement the BP algorithm using parallel computational approaches. BP has been implemented on Shared Memory Architecture (SMA) by Open Multiprocessing (OpenMP) [48], on Distributed Memory Architecture (DMA) by Message Passing Interface (MPI) [49] and on Graphics Processing Units (GPU) by Compute Unified Device Architecture (CUDA) [50][51].

2.2.3.3 Conjugate Gradient Training Algorithm

The conjugate gradient method is originally derived from quadratic minimization and the minimum of the objective function E can be efficiently found with N_w iterations [1]. With initial gradient $\boldsymbol{g}_{initial} = \left.\frac{\partial E_{Tr}}{\partial \boldsymbol{w}}\right|_{w=w_{initial}}$, and direction vector $\boldsymbol{h}_{initial}$ = -$\boldsymbol{g}_{initial}$, the conjugate gradient method recursively constructs two vector sequences,

$$\boldsymbol{g}_{next} = \boldsymbol{h}_{now} + \lambda_{now}\boldsymbol{H}\boldsymbol{h}_{now} \tag{2.11}$$

$$\boldsymbol{h}_{next} = -\boldsymbol{g}_{now} + \gamma_{now}\boldsymbol{h}_{now} \tag{2.12}$$

$$\lambda_{now} = \frac{\boldsymbol{g}_{now}^T \boldsymbol{g}_{now}}{\boldsymbol{h}_{now}^T \boldsymbol{H}\boldsymbol{h}_{now}} \tag{2.13}$$

$$\gamma_{now} = \frac{\boldsymbol{g}_{next}^T \boldsymbol{g}_{next}}{\boldsymbol{g}_{now}^T \boldsymbol{g}_{now}} \tag{2.14}$$

or,

$$\gamma_{now} = \frac{(\boldsymbol{g}_{next} - \boldsymbol{g}_{now})^T \boldsymbol{g}_{next}}{\boldsymbol{g}_{now}^T \boldsymbol{g}_{now}} \tag{2.15}$$

where $\boldsymbol{h}$ is called the conjugate direction and $\boldsymbol{H}$ is the Hessian matrix of the objective function E_{Tr}. λ and γ are called learning rate, and subscription *now*, *next* are the current and next values of $\boldsymbol{g}$, $\boldsymbol{h}$, λ and γ, respectively. Here, Equation (2.14) is called the Fletcher-Reeves Equation and Equation (2.15) is known as the Polak-Ribiere formula. To avoid the need of hessian matrix to compute the conjugate direction, we proceed from

$\boldsymbol{w}_{now}$ along the direction $\boldsymbol{h}_{now}$ to the local minimum of E_{Tr} at $\boldsymbol{w}_{next}$ through line minimization, and then get $\boldsymbol{g}_{next} = \left.\frac{\partial E_{Tr}}{\partial \boldsymbol{w}}\right|_{\boldsymbol{w}=\boldsymbol{w}_{next}}$. This $\boldsymbol{g}_{next}$ can be used as vector of Equation(2.11), and as such Equation(2.13) is no longer needed. We can make use of this line minimization concept to find conjugate direction in neural network training, thus avoiding intensive Hessian matrix computations. In this method, the descent direction is along the conjugate direction which can be accumulated without computations involving matrices. As such, conjugate gradient methods are every efficient and scale well with the neural network size.

2.2.3.4 Quasi-Newton Training Algorithm

Quasi-Newton algorithm is also derived from quadratic objective function optimization [1]. The inverse of Hessian matrix $\boldsymbol{B} = \boldsymbol{H}^{1}$ is used to bias the gradient direction. In Quasi-Newton training method, the weights are updated by

$$\boldsymbol{w}_{next} = \boldsymbol{w}_{now} - \eta \boldsymbol{B}_{now} \boldsymbol{g}_{now} \tag{2.16}$$

Standard Quasi-Newton methods require N_w^2 storage space to maintain an approximation of the inverse Hessian matrix, where N_w is the total number of weights in the neural network structure, and a line search is indispensable to calculate a reasonably accurate step length. A reasonable accurate step size is efficiently calculated in

one-dimensional line search by a seconder-order approximation of the objective function. Through the estimation of inverse Hessian matrix, Quasi-Newton has faster convergence rate than conjugate gradient method.

2.3 Overview of Parallel Computing Architectures

Traditional computer software has been written for sequential computation. An algorithm is implemented as a serial stream of instructions. These instructions are executed on a Central Processing Unit (CPU) on one computer. A target problem is solved by executing instructions on after another. Parallel computing techniques execute instructions simultaneously by multiple processing elements, which are accomplished by breaking the problem into independent parts so that each processing element can execute its own part concurrently with the others [52]. The processing elements include a variety of computational resources, such as a single computer with multiple processors, several interconnected computers, graphics processing units, or any combination of the above.

To determine the effect of parallel computing algorithm, we measure the parallel speed up against the corresponding sequential algorithm. Let T_1 be the time of executing algorithm sequentially. For parallel computing, let p be the number of processors, then the execution time of parallel algorithm with p processors is T_p. The speed up S_p is measured as

$$S_p = \frac{T_1}{T_p} \tag{2.17}$$

Ideally, a linear speed up $S_p = p$ is expected to obtain when using p processors. However, in most cases, the ideal speed up cannot be achieved due to the reason of design of the algorithm, competition of the shared memory, communication and synchronization, etc. Efficiency is introduced to show how the processors are well-utilized to execute the parallel algorithm. It also indicates how much effort is wasted in communication and synchronization between multiple processors. The efficiency of p processors is determined as

$$E_p = \frac{S_p}{p} \tag{2.18}$$

Both speed up and efficiency are targets of designing a parallel algorithm. Parallel computation can be performed on a various kinds of platforms classified by different parallel hardware architectures. Different kinds of Application Programming Interfaces (APIs) provide support for developing parallel applications on different parallel hardware architectures. Two hybrid parallel architectures and their programming models are reviewed below.

2.3.1 Hybrid Distributed-Shared Memory Architecture

The structure of Hybrid Distributed-Shared Memory Architecture (HDSMA) is

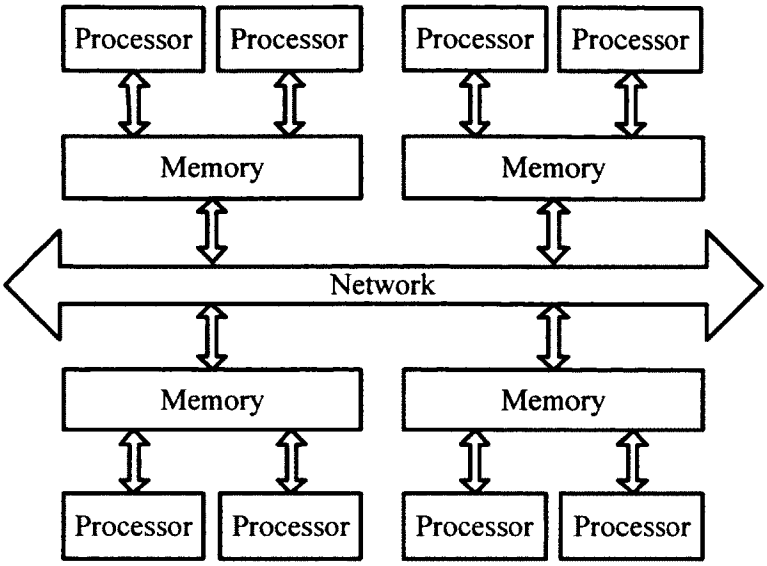


Figure 2.2 Structure of Hybrid Distributed-Shared Memory Architecture (HDSMA)

shown in Figure 2.2. In this architecture, the system consists of multiple interconnected computers called nodes. Each node has its private local memory that is uniformly shared by multiple processors. On one node, multiple processors can operate independently but share the same memory resources on that node. The communications between nodes are performed by message passing. The most typical network is Ethernet. HDSMA contains the advantage of Shared Memory Architecture (SMA) that the unified global memory address space provides a fast data sharing between tasks and a user-friendly programming perspective. This architecture exploits the maximum CPU capacity in the computer cluster. Compared with the SMA, HDSMA expands the performance on a single computer with SMA computer to a group of computers. Compared with Distributed

Memory Architecture (DMA), HDSMA makes full use of all available processors on each node and provides extra computational capability.

The programming model of HDSMA is the combination of programming models of SMA and DMA. On the top level, tasks are distributed among all available nodes in the same way as DMA by Message Passing Interface (MPI) [53], which is the only message passing library that is considered as a standard. On the lower level, the master thread on each node will fork a group of slave threads to execute tasks that distributed to that node simultaneously by Open Multiprocessing (OpenMP) [54]. An example of loop construct that utilizing hybrid MPI and OpenMP is shown below. Where m is the number of iterations of the loop, n is the number of nodes in the group connected by the communicator *comm.* p is the number of parallel threads on each node. Two different ways of data communication are implemented. Array a is broadcasted by node 0 to all other nodes by collective communication method. The loop construct is broken into multiple parts and distributed to all nods. Each node executes its own part of loop construct based on the identifier *rank* of the node. The result array b is collected by node 0 from all other nodes by point-to-point communication method. *MPI_Barrier* are used to ensure the synchronization of all nodes in the group [55].

HDSMA reaches the maximum speed up against sequential computation. It is a highly scalable architecture that allows user easily adding more nodes to the entire system to further increase the overall computational capability.

```
MPI_Status status;

MPI_COMM comm;

MPI_Init( &argc, &argv );

MPI_Comm_rank(comm, &rank);

//Data sent from process 0 to all other processes

MPI_Bcast(a , m, MPI_DOUBLE, 0, comm);

MPI_Barrier(MPI_COMM_WORLD);

#pragma omp parallel for private(i) num_threads (p)

for (i = n / rank; i < n / (rank + 1); i++)

    b[i] = a[i] * cos(i + 1);

MPI_Barrier(comm);

//Data received by process 0 from all other processes

if (rank ==0) {

    for (i = 1; i < m; i++)

        MPI_Recv(b + i * n / m, n / m, MPI_DOUBLE, i, 99, comm, &status);

}

else

    MPI_Send(b + rank * n / m, n / m, MPI_DOUBLE, 0, 99, comm);

MPI_Barrier(comm);

MPI_Finalize();
```

2.3.2 Hybrid Shared Memory-Graphics Processing Units Architecture

Graphics Processing Units (GPUs) are specialized circuits to accelerate building images for display. Compared with four or six processors contained in a main-stream Central Processing Unit (CPU), GPUs may have hundreds of processors called stream processors. These stream processors are formed in highly parallel structures. Modern GPUs have been successfully applied to processing extensive data computation in parallel, such as machine learning [56][57], numerical analytics [58], seismic modeling[59], etc.

Hybrid Shared Memory-Graphics Processing Units Architecture (HSMGPUA) is defined by multiple GPUs installed on one system with multiple processors. Each GPU is mapped to be controlled by one processor in the CPU. Figure 2.3 shows the structure of HSMGPUA architecture. Memory operations can be either between the system memory and GPU memory or between two GPU memory spaces. Having multiple GPUs on one system has many advantages. One advantage is that a parallel computational task can be broken into multiple portions and assigned by different GPUs to execute different portions simultaneously. In this method, data transfer between the system memory and the memory spaces on multiple GPUs can be performed concurrently. The quantity of data transferred to each GPU can be reduced to be one out of number of GPUs. The influence of data transferring overhead can be significantly decreased. Meanwhile, data transfer between multiple GPUs is extremely fast since multiple GPUs are directly linked

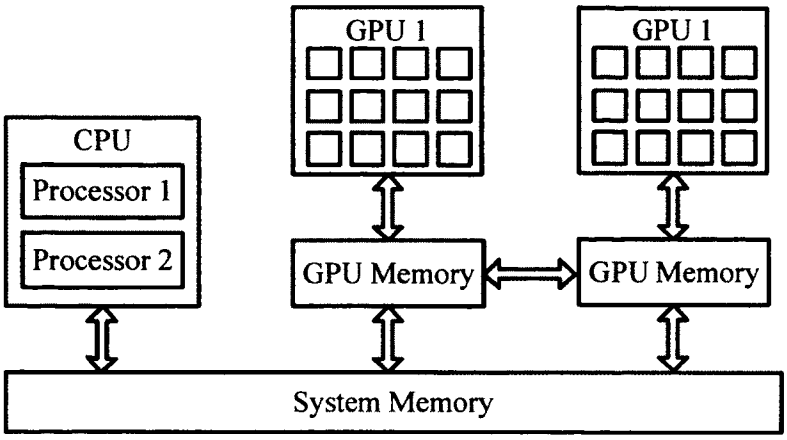


Figure 2.3 Structure of Hybrid Shared Memory-Graphics Processing Units Architecture

by the same high speed PCI-E bus. On the other hand, different GPUs can be assigned to execute totally different parallel computational tasks. Since different GPUs are controlled by different processors, a pair of processor and GPU can be treated as an individual system.

The programming model of HSMGPUA is the combination of programming models of SMA and General-Purpose computing on Graphics Processing Units (GPGPU). On the top level, multiple processors are created by OpenMP to bind different GPUs. On the lower level, parallel computational tasks are executed by GPGPU programming models. The synchronization of GPU stream processors is performed by GPGPU, while the synchronization of multiple GPUs is performed by OpenMP.

There are two major programming interface models available for GPGPU. The most

universal one is Open Computing Language (OpenCL), which is maintained by the non-profit technology consortium *Khronos Group* [60]. OpenCL is an open standard that provides application access to GPGPU by both task-based and data-based parallelism. OpenCL supports various kinds of CPUs, GPUs and even FPGA. Another model is Compute Unified Device Architecture (CUDA) developed by *Nvidia* [61]. CUDA is a computing engine that enables parallel computation on *Nvidia*'s own brand GPU. CUDA provides a low level driver API and a higher level runtime API. The driver API is similar to OpenCL in which users are fully responsible for controlling over the hardware. The runtime API provides a C-like set of routines and extensions and hides detailed hardware implementation for users. Since CUDA runtime API provides users an easy way to develop GPGPU programming, our examples will be implemented by CUDA runtime API.

Chapter 3

Parallel Automatic Data Generation

3.1 Introduction

Training data generation is one of the major stages in neural network model development. Training data can be obtained from measured or simulated microwave data. Typical examples of software simulators are *Ansoft HFSS* [62], *Agilent ADS* [63], *CST Microwave Studio* [64], etc. The conventional method to obtain training data is to manually change the physical/geometrical parameters after every simulation to obtain new outputs. This process can easily bring about errors when large amounts of training data are required. The Automatic Data Generator (ADG) has been developed to automate the training data generation process to minimize manual work and the chances of human errors [65]. Since the physical/EM simulations are expensive, there is high demand on the acceleration on training data generation stage. In the Parallel Automatic Model Generation (PAMG) technique, training data generation has been parallelized on multiple processors on one computer by simultaneously driving multiple simulators on multiple processors [15]. However, with increased complexity in the structures of microwave devices and circuits, the number of training data required for developing an accurate neural network model is increasing. Meanwhile, with the development of computer

technology, it is easy and affordable to set up a parallel computational environment that consists of multiple computers with multiple processors on each computer. The parallelization of training data generation can be further expanded to be executed on multiple computers.

3.2 Key Aspects of Parallel Automatic Data Generation

The proposed Parallel Automatic Data Generator (PADG) technique is an enhanced training data generator implemented on the Hybrid Distributed-Shared Memory Architecture (HDSMA), i.e., a cluster consists of several network connected computers with local memory on each computer shared by several processors. PADG can be integrated into PAMG to further reduce data generation time; it can also be invoked by other neural network algorithms, such as neural network model optimization algorithm [67], to efficiently generate training/testing data. Figure 3.1 shows the framework of PADG algorithm. During neural network model development or optimization, user's program will send a request to PADG to generate more data. The node, on which user's program is running, is numbered as node 1. Node 1 has a shared folder that can be accessed by all the nodes in the cluster; it is used for data communication in the distributed memory system. Each node has a local folder that can only be accessed by that node; it is used for data storage in the shared memory system. Once PADG receives the request, it will copy design files from user's working folder to the shared folder on the node 1. The design file created by simulator contains physical/geometrical values, boundary conditions, frequency sweep parameters, etc. Meanwhile, physical/geometrical

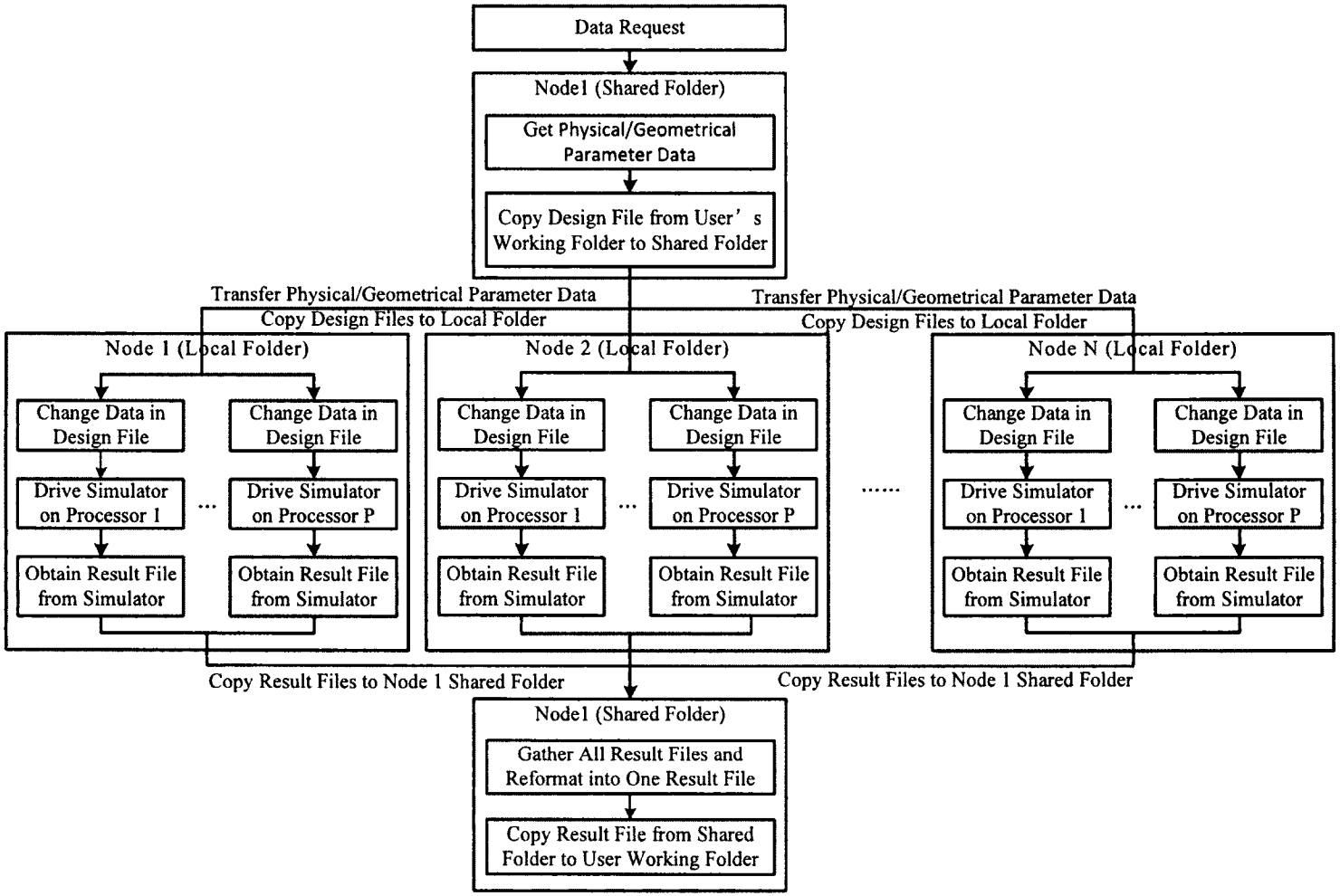


Figure 3.1 Framework of Parallel Automatic Data Generation (PADG) on hybrid distributed-shared memory system

parameter data is transferred among node 1 and all other nodes. Design files will be duplicated into several separate copies from shared folder to each node's local folder. Once each node, design files will be updated with new physical/geometrical parameters. Multiple simulators are driven concurrently on multiple processors on each node. After all simulations are finished, the result files are generated by the simulators and transferred from the local folder on each node to the shared folder on node 1, where they are then combined and converted into the format that user's program can understand.

PADG is designed to achieve maximum speed up for training data generation process. It takes full advantage of HDSMA to improve the efficiency of using computer resources. Furthermore, PADG is fully automatic such that human labor and chance of error would be significantly reduced, especially when large amounts of training data are requested to be generated. PADG is also a scalable and portable tool to run on any computer or cluster. When PADG is implemented on shared memory system, i.e., a single computer, it operates in the same way as the training data generator in PAMG. User can also add more computer resources, i.e., adding more CPUs on one computer or adding more nodes in the cluster, to increase the computing capability of PADG. PADG requires a setup file that contains resource information as shown below.

```
24
qjzhpcnode1001      0
qjzhpcnode1002      4
qjzhpcnode1003      0
```

PADG is a systematic tool that can distribute tasks properly based on the number of nodes, the number of processes on each node and the number of licenses of simulation tools, which are demonstrated in the setup file. The first line of the setup file indicates the number of licenses of simulation tools; it determines the maximum number of parallel processors to be used in the cluster. It can be set to 0 if there is no license limitation. Starting from the second line, the name of computer node and the number of processors to be used on that node is indicated. User can assign number of processors on each node based on specific computer resources. Tasks are automatically distributed based on the information in the setup file provided by the user. Each node will be assigned the same or similar number of tasks to balance the overall workload. If tasks cannot be evenly distributed on all nodes, extra tasks will be assigned to the nodes starting from the lowest node number. The scalability of PADG provides user a simple way to add or reduce computer resources by adding or removing lines of computer nodes in the setup file and/or changing the number of processors to be used on the nodes.

3.3 Task Distribution Strategy

Task Distribution is very important to the performance of parallel computation. The total execution time for a parallel program implemented on the Hybrid Distributed-Shared Memory Architecture (HDSMA) depends on the maximum of each node's execution time. If tasks are not properly distributed to the nodes, some nodes will get much more tasks than the other nodes and finish executing these tasks with much longer time than the other nodes. The training data of neural networks is often obtained by commercial simulation software and the licenses of commercial simulators are often

expensive and limited. The license distribution should also be considered as part of task distribution. The target of task distribution is to balance the work load on each node. Since each license is mapped to one processor for execution, the major objective of task distribution is to determine the number of tasks distributed to each processor and the number of processors to be invoked on each node.

Consider a cluster consists of multiple nodes with multiple processors on each node. Let M be the number of available licenses, N be the number of computer nodes and K be the number of tasks to be distributed. The first step is to properly distribute all tasks to each processor. Let $\boldsymbol{P} = [P_1, P_2, \cdots, P_M]$ be a vector representing the number of tasks distributed to each processor. Each processor is designed to get equal or similar number of tasks to minimize the number of iterations for M processors to execute K tasks. If K tasks cannot be evenly distributed to M processors, extra jobs will be added to the processors starting from the lowest processor number. The number of tasks distributed to the i^{th} processor can be determined as

$$P_i = \begin{cases} K \div M, & K \mid M \\ \lfloor K \div M \rfloor + 1, & i \le (K \bmod M), \quad K \nmid M \\ \lfloor K \div M \rfloor, & i > (K \bmod M), \quad K \nmid M \end{cases} \tag{3.1}$$

where $K \mid M$ means K can be divided exactly by M, $K \nmid M$ means K cannot be divided exactly by M, $K \bmod M$ means the reminder of K divided by M and $\lfloor K \div M \rfloor$ means the round off number of K divided by M.

The next step is to properly distribute all processors to each node. Let $\boldsymbol{Q} = [Q_1, Q_2, \cdots, Q_N]$ be a vector representing the number of processors to be executed on each

node. Each node is designed to invoke equal or similar number of processors to minimize the influence of the competition for the shared processor-memory path by multiple processors on that node. The processor distribution is quite similar to the task distribution. The number of processors invoked on the j^{th} node can be determined as

$$Q_j = \begin{cases} M \div N, & M \mid N \\ \lfloor M \div N \rfloor + 1, & j \le (M \bmod N), \quad M \nmid N \\ \lfloor M \div N \rfloor, & j > (M \bmod N), \quad M \nmid N \end{cases} \tag{3.2}$$

The final step is to distribute all the tasks to all available nodes so that the data communication by Message Passing Interface (MPI) can follow this strategy and properly transfer simulation design files and physical/geometrical parameter data to each node. Let $\boldsymbol{T} = [T_1, T_2, \cdots, T_N]$ be a vector representing the number of tasks to be distributed to each node. The number of tasks distributed to the j^{th} node is the summation of the number of tasks to be executed on all processors on the j^{th} node as

$$T_j = \sum_{i=k}^{k+Q_j} P_i \tag{3.3}$$

where k is the index of starting number of processor on the j^{th} node in the cluster group.

When the task distribution is finished, PADG can then distribute T_j tasks to the j^{th} node by MPI, transfer the corresponding design files and parameter data, and invoke Q_j processors on the j^{th} node to run simulations simultaneously by OpenMP. After all simulations are finished, PADG will collect results files from each node based on the same task distribution strategy.

3.4 Verification of Parallel Automatic Data Generation

In order to verify the PADG algorithm, we use an interdigital band-pass filter example as illustrated in Figure 3.2. The filter example is constructed by *Ansoft HFSS* EM solver for the ease of use and the fast speed of *Ansoft HFSS*.

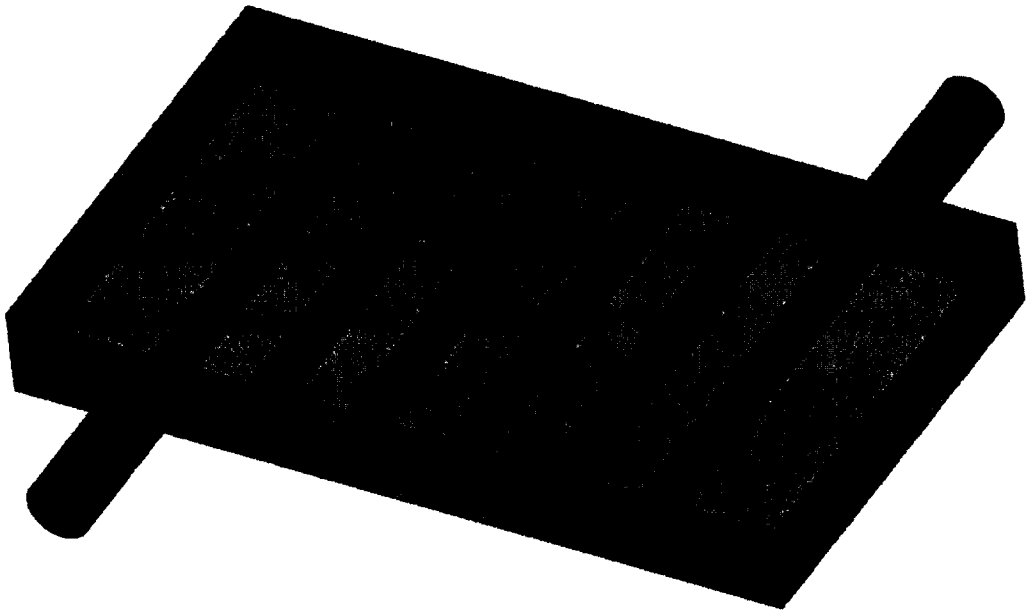


Figure 3.2 Structure of interdigital band-pass filter with four design variables

The first step in PADG is to create design files using *HFSS*. We can draw the structure of the band-pass filter; define the kind of materials to be used, the boundary and feed port conditions, the frequency sweep ranges and the units of measurement, etc. Once all the information above is defined, a "*.hfss*" file will be saved as one design file. HFSS uses scripting languages to support easy and automatic driving of EM simulator. Frequently used geometries can be recorded and generated repeatedly by running a script.

EM outputs like S-parameter can also be recorded in the script to generate output files by driving the EM simulator. The Visual Basic script is recorded within the HFSS design environment and saved as a "*.vbs*" file. Any behavior performed in the HFSS GUI can be recorded to the "*.vbs*" script and can be reproduced by running the recorded scripts. The "*.hfss*" file along with "*.vbs*" file provides user full controls of driving HFSS simulator.

The next step is to generate a setup file for PADG. The setup file contains information of the location of design files, the desired location of output files and the name and value of parameters to be simulated. PADG will first copy the design files to the shared folder, then determine the number of simulations, distribute the workload and transfer parameter names and new values to all available nodes. Each node will get several copies of design files and save in the local folder. The design files are renamed and numbered in sequence for each simulation. For each simulation, PADG will search the name of parameter in "*.vbs*" file and change the value of that parameter with new value defined in the setup file. PADG will also search and modify the name and/or path of "*.hfss*" files and "*.csv*" output files to ensure the HFSS driven on different processors solve different projects and save the result to the corresponding files.

After all the "*.vbs*" files are updated, PADG will drive the HFSS simulator to execute the scripts simultaneously on the available processors across the available nodes. When each simulation is solved, HFSS will save the output result into a "*.csv*" file. When all simulations are finished and all "*.csv*" files are generated in the local folder on all nodes, the "*.csv*" files will be copied to the shared folder on node 1. PADG will then extract, reformat and combine the data stored in the "*.csv*" files and save it as "*.dat*" file that can be recognized for neural network model training.

The following "*BPF_Optimization_final.vbs*" file is used to illustrate the idea of updating file with information provided by the user. The first line indicates the location of HFSS project file to be opened. PADG will search for the "oDesktop.OpenProject" and change the file path between the quotation marks. The second line indicates the project name to be simulated, which should be the same as the project file name. PADG will search for the "oDesktop.SetActiveProject" and change the active project name between the quotation marks. Parameter values are changed in the following "oDesign.ChangeProperty" sections, where PADG searches for the name of parameter as "L", "S1", "S2" and" S3", then replaces the existing value with new value corresponding to that parameter. The "oDesign.AnalyzeAll" section drives HFSS simulator and the "oModule.CreateReport" section creates a plot that contains frequency as the X axis value and the real and imaginary part of S11 and S12 as the Y axis values, respectively. Result data are exported by the "oModule.ExportToFile" section. The content between the following quotation marks indicates the location of output ".csv" file. PADG will search for the "oModule.ExportToFile" then modify the ".csv" file location information.

```
...

oDesktop.OpenProject "C:\BPF_Optimization_final.hfss"

Set oProject = oDesktop.SetActiveProject("BPF_Optimization_final")

Set oDesign = oProject.SetActiveDesign("BPF3_o")

oDesign.ChangeProperty ... Array("NAME:ChangedProps", Array("NAME:L",
"Value:=", "3.5mm"))))

 oDesign.ChangeProperty ...Array("NAME:ChangedProps", Array("NAME:S1",
"Value:=", "1mm"))))

 oDesign.ChangeProperty ... Array("NAME:ChangedProps", Array("NAME:S2",
"Value:=", "2.3mm"))))

oDesign.ChangeProperty ...Array("NAME:ChangedProps", Array("NAME:S3",
"Value:=", "2.6mm"))))

oProject.Save

oDesign.AnalyzeAll

Set oModule = oDesign.GetModule("ReportSetup")

oModule.CreateReport "XY Plot 1", ...Array("X Component:=", "Freq", "Y
Component:=", Array("re(S(1,1))", "im(S(1,1))",  "re(S(1,2))", "im(S(1,1))")), Array()

oModule.ExportToFile "XY Plot 1", "C:\BPF_Optimization_final.csv"
```

Parameter	L	S1	S2	S3
Minimum Value (mm)	3.0	0.8	2.2	2.6
Maximum Value (mm)	4.5	1.1	2.4	2.8
Step size (mm)	0.5	0.1	0.1	0.1
Quantity	4	4	3	3

Table 3.1 Geometric value of design parameters of interdigital band-pass filter

In order to demonstrate the advantages of PADG, we executed training data generation on shared memory system, distributed memory system and hybrid distributed-shared memory system to compare the performance of PADG on the above three platforms. As illustrated in the "*BPF_Optimization_final.vbs*" file, four geometrical parameters are swept, which are L, S_1, S_2 and S_3. Table 3.1 shows the minimum value, maximum value and step size for these four parameters. The step size of each design parameter is determined by the sensitivity of that parameter to the output response. We choose a relatively large step size for the parameters with low sensitivity and relatively small step size for the parameters with high sensitivity. The sweep frequency is from 0.6GHz to 2.4GHz with a step size of 4MHz. The total number of simulation is 4*4*3*3=144. First the PADG is executed to drive multiple processors to run these 144 simulations on one computer. We keep only one line of computer nodes in the setup file for PADG and change the number after the node name to measure the simulation time of utilizing different number of processors. Then more nodes are added to the setup file and

keep the number after each node to be one to measure the simulation time of implementing PADG on distributed memory system. Then we change the number of processors for each node and drive PADG with different combinations of the number of nodes and the number of processors on each node. The PADG is implemented on a cluster consists of nine computer nodes. Each node is equipped with two quad-core Intel Xeon E5640 processors with hyper-threading feature providing sixteen processing threads on each node.

Number of Threads	1	2	3	4	6	8	9	12	16
Time (min)	352.2	181.1	128.9	101.4	76.3	64.3	60.4	54.0	49.1
Speed Up	N/A	1.95	2.73	3.47	4.62	5.48	5.83	6.53	7.17
Efficiency (%)	N/A	97.27	91.10	86.82	76.96	68.49	64.75	54.39	44.83

Table 3.2 Speed up and efficiency of executing Parallel Automatic Data Generation on Shared Memory Architecture

Table 3.2 shows the execution time, speed up and efficiency of implementing PADG on a shared memory system. We can see a nonlinear growth in speed up with the increased number of threads, which leads to a continuous drop in the efficiency. The tendency is that if one more thread is added, an average of 4% efficiency will be lost. This is caused by the competition of multiple threads to the shared path between the

processors and the system memory, which is the major disadvantage of shared memory architecture. Another disadvantage is that the shared memory capacity may not be enough if we drive multiple simulators to solve examples with complex structures. One thread has to wait for other threads releasing some memory space, which will cause time wasted on thread idling and even simulator error or failure.

PADG on SMA is the simplest way to achieve speed up for training data generation. It requires minimum cost on computer hardware since nowadays most CPUs have multiple processors that can directly execute PADG for parallel computation. Although the efficiency is reduced with more threads, the overall speed up is keeping increased. Large amounts of CPU time on training data generation could be easily saved by applying PADG on a single computer.

Number of Nodes	2	3	4	6	8	9
Time (min)	176.91	118.24	88.91	59.44	44.66	39.90
Speed Up	1.99	2.98	3.96	5.93	7.89	8.83
Efficiency (%)	99.55	99.30	99.04	98.75	98.58	98.08

Table 3.3 Speed up and efficiency of executing Parallel Automatic Data Generation on Distributed Memory Architecture

There are many advantages of distributing parallel tasks on multiple computers. From Table 3.3, we can see that the efficiency is always nearly 100% while increasing the number of computer nodes in the parallel task distributing. The average cost of adding one more node to the distributed system is only 0.28% in the overall efficiency. The overhead of the distributed system is the time elapsed on data transferring, which includes copying the design files and the output files, and transferring physical/geometrical parameter data. All nodes are connected by a 10Gbps Ethernet that provides very high data transfer speed. Adding more nodes will only slightly increase time on data transfer. There is only one thread invoked on each node, which means there is no competition for shared memory and no efficiency lost on a single node. Another advantage of distributed memory architecture is that the memory usage on a single node is the same as invoking one thread on shared memory architecture, so that the risk of the application error or failure could be minimized or avoided for examples with complex structures.

We can draw a conclusion that PADG on DMA achieves extremely high speed up with nearly full efficiency. Compared with PADG on SMA, the speed up of distributing parallel task on eight individual nodes is even higher than the speed up of invoking sixteen threads on a single computer. In other words, eight licenses could be saved and more speed gain is obtained. Since license is usually a high cost for commercial simulation tools, users can have the choice to save the budget on simulation tool license with the efficient usage of their distributed memory system.

In this example, a total number of 24 licenses for *Ansoft HFSS* is available. We have nine computer nodes with sixteen threads on each node. PADG on neither SMA nor

Number of Nodes * Number of Threads Per Node	2 * 12	3 * 8	4 * 6	6 * 4	8 * 3	6 * 3 + 3 * 2
Total Number of Threads	24	24	24	24	24	24
Time (min)	27.09	21.64	19.73	17.15	16.28	16.15
Speed Up	13.00	16.28	17.85	20.53	21.63	21.81
Efficiency (%)	54.18	67.83	74.39	85.56	90.14	90.86

Table 3.4 Speed up and efficiency of executing Parallel Automatic Data Generation on Hybrid Distributed-Shared Memory Architecture

DMA can make full use of all available *HFSS* licenses. By executing PADG on a hybrid distributed-shared memory system, we are able to use all available 24 licenses. Table 3.4 shows six different combinations of the number of nodes and the number of threads used for each node. We can see that the overall speed up varies widely with different configurations. If we take a close look into the overall efficiency with regard to the number of threads used on each node, the efficiency is similar to the efficiency with the same number of threads on SMA as shown in Table 3.2. The loss in the efficiency is the combination of time elapsed on the processor-memory path competition on SMA and time elapsed on data transfer on DMA. We propose to distribute parallel tasks as much as possible to all available nodes. Within each node, the number of threads used can be symmetric or asymmetric based on the total number of simulator licenses. PADG will automatically distribute tasks and determine the number of threads used on each node based on the information provided by the user in the setup file as described in Section 3.3.

In conclusion, PADG on HDSMA makes full use of available simulator licenses to achieve maximum speed up. A speed up of 21.81 and efficiency of 90.86% was achieved by using nine nodes with 24 simulator licenses. Higher speed up can be expected if more nodes are added.

3.5 Summary

Parallel Automatic Data Generator (PADG) is a powerful tool for training data generation. PADG runs on a hybrid distributed-shared memory system and provides maximum speed up based on the available resources. PADG is a scalable model that allows user to add more computer resources to improve the computational capability. PADG is a highly systematical model that automatically distributes parallel tasks to all available nodes.

PADG can significantly reduce the time required for data generation process, which includes training data generation, validation data generation and testing data generation in various kinds of neural network applications. PADG is also a universal model that can drive multiple kinds of software simulators based on the information provided by the user.

Chapter 4

Parallel Multiple ANN Training

4.1 Introduction

Parallel Artificial Neural Network (ANN) training has been achieved on various kinds of architectures [48][49][50][51]. However, these implementations are focused on a single neural network structure. With the increased complexity of microwave/RF structures, the number of design variables is keeping increasing. The amount of neural network training data increases fast with the number of input neurons. Several advanced neural network technologies have been introduced to reduce the quantity of training data required to develop an accurate neural network model [12][13][14]. One study is known as Modular Neural Network (MNN) [17][18]. MNN consists of multiple independent neural networks. Each neural network operates as a module and provides independent outputs based on separate inputs. With the exploitation of structural decomposition [67][68], a complex microwave/RF structure can be decomposed into multiple separate parts. Each part is modeled independently by neural network as a module of MNN. Other additional neural networks may also be included in the MNN to learn the nonlinear relationship between the inputs of separate models and the coefficients of frequency mapping, etc. The conventional method is to train these modules one and after another, which requires a lot of human labor and has large chances of human error, especially

when large numbers of modules are presented. For a single module, parallel techniques can be applied to accelerate the neural network training process. However, after neural network training for one module is finished, the user has to manually define neural network structure for another module. The reason is that different models may have different number of inputs, hidden neurons and outputs due to the specific structural decomposition method. In other words, different models may have different neural network structures. The neural network structure creation is repetitive and time-taking process. The conventional method to train multiple neural network models is not very efficient. There is growing need to have a universal method to train multiple neural networks automatically and simultaneously.

4.2 Parallel Multiple ANNs Training on Central Processing Unit

Parallel Multiple ANNs Training on Central Processing Unit (PMAT-C) is proposed to training multiple ANNs on the Shared Memory Architecture (SMA), i.e., a single computer with multiple processors, or on the Hybrid Distributed-Shared Memory Architecture (HDSMA), i.e., a cluster consists of multiple computers with multiple processors on each computer. In contrary to the existing parallel techniques applied to the ANN training, PMAT-C does not break the structure of a single ANN model. PMAT-C will execute ANN training for one ANN model completely on one processor to minimize data exchange. Multiple ANN models are trained concurrently on multiple processors.

Figure 4.1 shows the framework of PMAT-C on a HDSMA system. The framework is similar to the Parallel Automatic Data Generation (PADG) as shown in Figure 3.1.

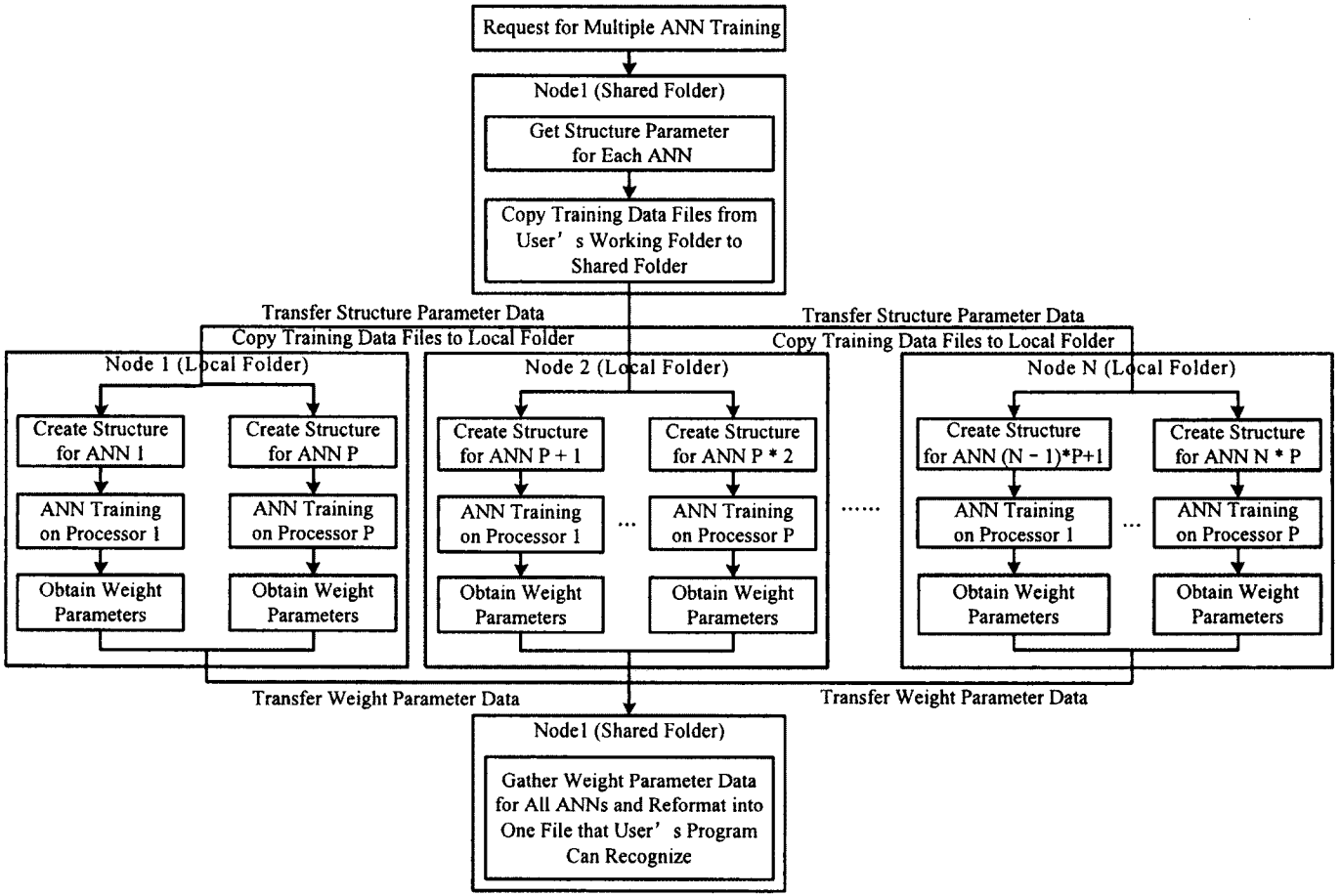


Figure 4.1 Framework of Parallel Multiple ANNs Training on Central Processing Unit (PMAT-C) on Hybrid Distributed-Shared Memory Architecture (HDSMA)

The benefit of have the similar framework of PMAT-C and PADG is that each processor in the HDSMA system can be assigned to generate training data and then process ANN training for one neural network model, i.e. one neural network module of the entire MNN model. The data exchanging can be minimized and the speed up and efficiency of building a MNN model could be significantly improved. Meanwhile, PADG and PMAT-C can also work separately based upon user's demand.

When PMAT-C is requested for multiple ANN training, it requires a setup file containing information about neural network structure, i.e. the number of inputs, the number of hidden neurons and the number of outputs, and the corresponding training data file path for each ANN. PMAT-C will check for the number of columns of training data for each training data file. If the number of columns of training data doesn't match with the summation of the number of inputs and the number of outputs, PMAT-C will give user a message and then terminate. After verifying all training data files are in the correct format, PMAT-C will send the ANN structure information and copy the training data files to each node by Message Passing Interface (MPI). On each node, PMAT-C will invoke multiple processors to train multiple ANNs simultaneously by OpenMP. The ANN training process can either be done by the implementation of classic training algorithms or by invoking neural network modeling software such as *NeuroModeler Plus* [69]. Since the ANN structures and number of training samples may have great difference by different ANN, the training time for each ANN may differ greatly. PMAT-C ensures synchronization after the completion of last ANN training. If ANN training is executed by classic training algorithms, the weights for each ANN will be transferred to node 1 and be reformatted to one file that user's program can recognize. If ANN training is

executed by invoking the network modeling software, the corresponding design file generated by the network modeling software will be copied to the shared folder on node 1, where user can make future use of these design files.

In contrary to the commercial simulation software to be invoked in PADG, the implementation of classic neural network training algorithm does not require licenses. PMAT-C can make full use of computer resources to achieve maximum speed ups. If the number of parallel tasks, i.e. the number of ANNs to be trained, is larger than the total quantity of processors in the cluster group, extra job will be added to the processor starting from the lowest node number. Let N be the number of nodes, M be the number of processors on each node and K be the number of tasks, i.e. the number of ANNs to be trained. Let $P = [P1, P2, ..., P_{M*N}]$ be a vector representing the number of tasks distributed to each processor. The number of tasks distributed to i^{th} processor can be determined as

$$P_i = \begin{cases} K \div (M \times N), & K \mid (M \times N) \\ \lfloor K \div (M \times N) \rfloor + 1, & i \le (K \bmod (M \times N)), \quad K \nmid (M \times N) \\ \lfloor K \div (M \times N) \rfloor, & i > (K \bmod (M \times N)), \quad K \nmid (M \times N) \end{cases} \tag{4.1}$$

where $K \mid (M * N)$ means K can be divided exactly by the product of M and N, $K \nmid (M * N)$ means K cannot be divided exactly by the product of M and N, K mod M means the reminder of K divided by the product of M and N and $\lfloor K \div (M * N) \rfloor$ means the round off number of K divided by the product of M and N.

Let $\boldsymbol{T} = [T_1, T_2, \ldots, T_N]$ be a vector representing the number of tasks to be distributed to each node. The number of tasks on j^{th} node is the summation of the number of tasks to be executed on all processors on j^{th} node as

$$T_j = \sum_{i=k}^{k+M} P_i \tag{4.2}$$

where k is the index of starting number of processor on j^{th} node in the cluster group. Data transfer between the node 1 and the other nodes can follow this strategy to properly distribute parallel tasks and collect training results.

4.3 Parallel Multiple ANNs Training on Graphics Processing Units

Back Propagation (BP) training algorithm has been implemented on Graphics Processing Units (GPU) by using Compute Unified Device Architecture (CUDA) model with its math library cuBLAS [50].

We use a simple matrix multiplication example $\boldsymbol{C} = \boldsymbol{A} * \boldsymbol{B}$ to illustrate the performance of CUDA and cuBLAS. Matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ both have the dimension of $N * N$, which gives $\boldsymbol{C}$ the same dimension as $N * N$. The classic algorithm is implemented by looping the elements in $\boldsymbol{A}$ and $\boldsymbol{B}$ and save the summation to the corresponding element in $\boldsymbol{C}$. An advanced algorithm called Basic Linear Algebra Subprograms (BLAS) provides optimized performance for vector operations, matrix-vector operations and matrix-matrix operations. There are many implementations of BLAS such as LAPACK[70], APPML[71], Intel MKL[72], etc. Nvidia has its own BLAS library called cuBLAS[73]

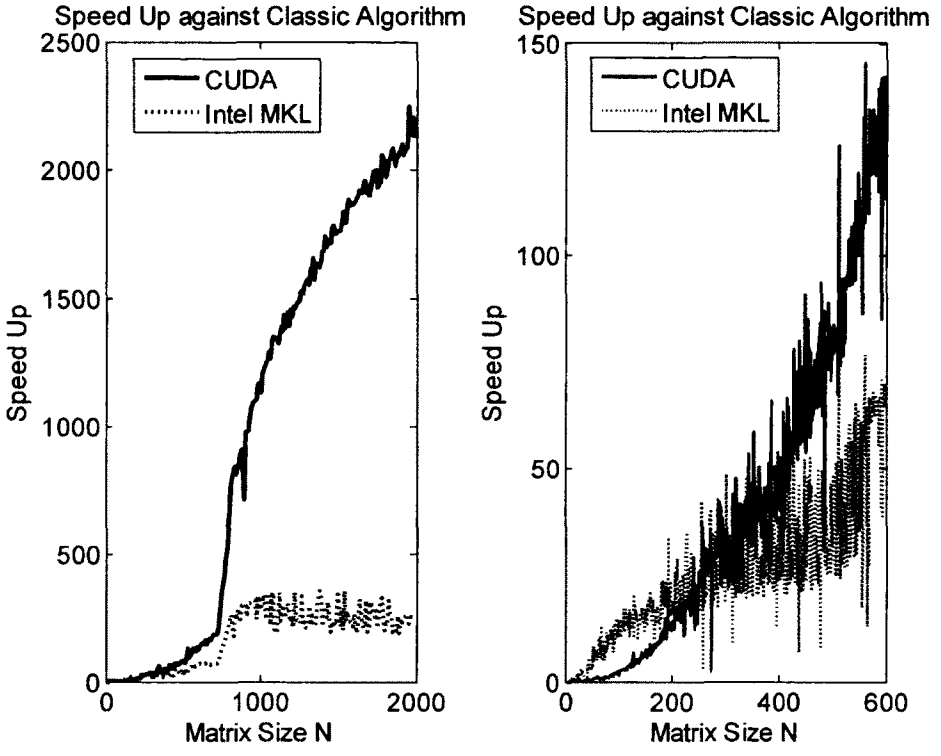
that preforms GPU-accelerated basic linear algebra operations. Our matrix multiplication example on GPU is implemented by cuBLAS and CUDA. To fairly compare the best result of speed up the can be achieved by optimized BLAS and parallel computation on CPU, we also implemented the matrix multiplication by Intel MKL, which can be easily set to support multi-processor parallelization on shared memory system.

The first step of CUDA program is to allocate memory space for ***A***, ***B*** and ***C*** on the GPU memory. These spaces are different than system memory space. So we use A_Dev, B_Dev and C_Dev as pointers to indicate that these spaces are on the GPU. The second step is to transfer the matrix data of ***A*** and ***B*** from the system memory to the GPU memory. The next step is to perform matrix multiplication. cuBLAS provides a simple routine to called *cublasSgemm* to perform matrix multiplication $\boldsymbol{C} = \alpha * \boldsymbol{A} * \boldsymbol{B} + \beta * \boldsymbol{C}$.

```
cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, N, N, N, 1.0, A_Dev,
N, B_Dev, N, 0.0, C_Dev, N);
```

where the matrix row size and column sizes are both N, $\alpha = 1.0$ and $\beta = 0.0$, which defines the function of matrix multiplication to be $\boldsymbol{C} = \boldsymbol{A} * \boldsymbol{B}$. This routine is similar to Intel MKL's matrix multiplication routine as *cblas_sgemm*. The final step is to transfer the result matrix ***C*** from GPU memory to the system memory and release the memory space on GPU. This example is implemented on a Quad-core CPU i7 920 with hyper-threaded for 8 parallel threads. GPU used in this example is Nvidia GTX 570. The matrix

size N is swept from 1 to 2000. Figure 4.2 shows the speed up for matrix multiplication by CUDA and Intel MKL against classic algorithm for different matrix sizes.



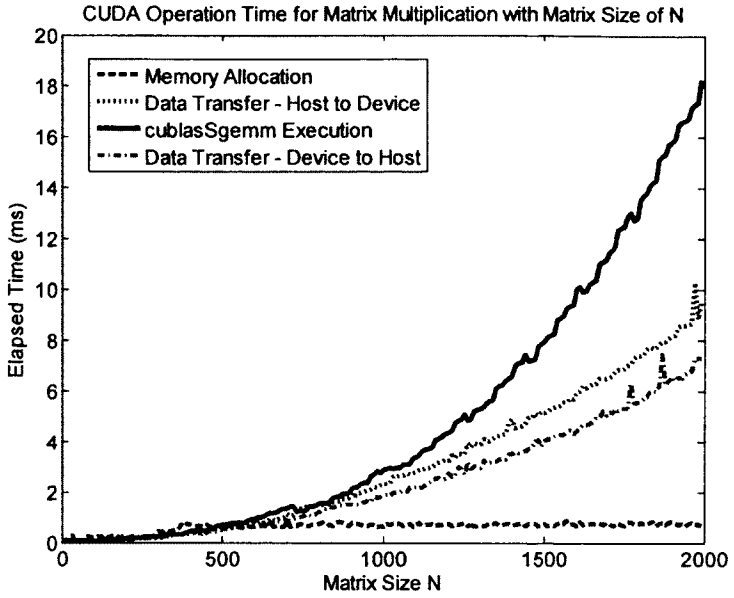
(a) N from 1 to 2000 (b) N from 1 to 600

Figure 4.2 Speed Up for matrix multiplication by CUDA and Intel MKL

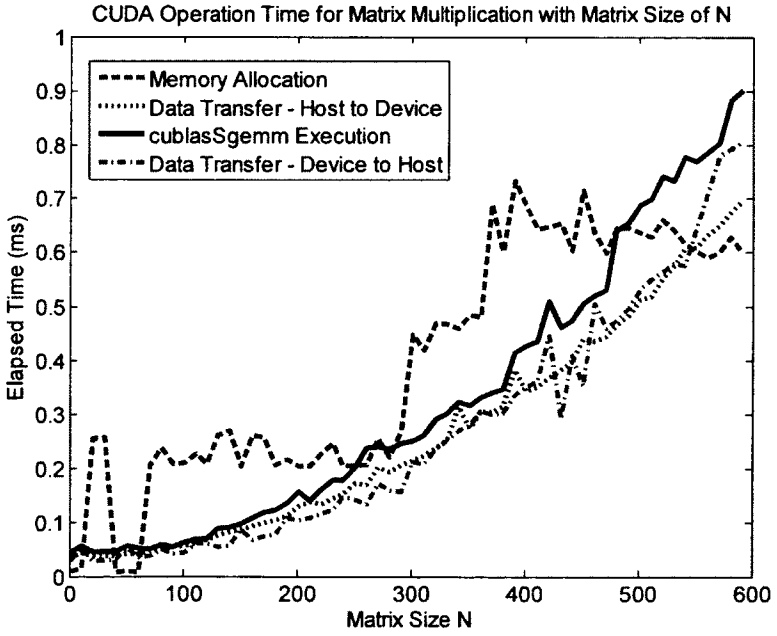
From Figure 4.2 (a), we can see that the speed up of CUDA keeps increasing with the larger matrix size, while the speed up of Intel MKL performed on eight parallel threads becomes constant when the matrix size reaches 800. Finally, when the matrix size reaches 2000, Intel MKL provides a speed up of 240 and CUDA performs nearly 2200

times faster than the classic algorithm on the CPU. It can be expected that CUDA can achieve higher speed up for larger matrix size while the speed up of Intel MKL remains near 240, which is considered to be the maximum speed gain by Intel MKL. However, if we take a close look to the speed up with relatively small matrix size shown in Figure 4.2 (b), we notice that CUDA does not always run faster than Intel MKL. This is because of the time consumption of memory allocation on GPU and data transfer between GPU memory and the system memory are inevitable. Figure 4.3 shows time for GPU operations in millisecond.

Compare Figure 4.3 (a) with Figure 4.3 (b), time for memory allocation is significant when the matrix size is relatively small. When the matrix size is larger than 400, time for memory allocation remains constant regardless of matrix size. We can see that time for data transfer between GPU memory and system memory is very close to *cublasSegmm* execution time when the matrix size is relatively small. Meanwhile, data transfer time is less than half of the *cublasSegmm* execution time when the matrix size is relatively large. The ratio of *cublasSegmm* execution time with memory operation time keeps decreasing with the increase of matrix size. The reason why CUDA runs slower than Intel MKL for small matrix size is that the memory operations occupy the most time of total CUDA execution. Since the classic algorithm and Intel MKL algorithm operates directly in the system memory, the memory operations between the system memory and GPU memory by CUDA are considered as overheads.



(a) N from 1 to 2000



(b) N from 1 to 600

Figure 4.3 CUDA operations time for matrix multiplication with matrix size of N

We can get an idea that in order to minimize the influence of the overheads, frequently transferring data between system memory and GPU memory should be avoided. In other words, we should once transfer all the data for parallel computation from system memory to GPU memory, perform computations on GPU as much as possible and only transfer the result data back from GPU memory to system memory.

There are many improvements to make based on the introduced implementation of BP algorithm on GPU. The effect of bias should be taken into account for the BP training algorithm. We should create larger matrices and perform computation on GPU as much as possible to minimize the influence of data transfer between system memory and GPU memory.

We propose to use the batch mode update method for BP training algorithm by developing multiple ANNs with the same weights for Parallel Multiple ANNs Training on Graphics Processing Units (PMAT-G). The number of ANNs is the same as the number of training samples. The proposed method can be implemented by converting Equation (2.1) to Equation (2.10) into matrix forms. Let n be the number of inputs, p be the number of hidden neurons, m be the number of outputs and s be the number of training samples. In order to take bias of hidden neurons into account, we can add a fictitious input neuron with value to be 1 into the input data of s training sample. Thus, we can create an input matrix $\boldsymbol{X}$ with dimension of $s * (n+1)$ to store all inputs data in s training samples along with the fictitious input neurons. The values of the elements in the $(n+1)^{th}$ column of matrix $\boldsymbol{X}$ are one. The weight matrix between the input layer and the hidden layer $\boldsymbol{W}^2$ will have the dimension of $(n+1) * p$ with all elements in $\boldsymbol{W}^2$ are random values representing initialize guess. The hidden matrix $\boldsymbol{Z}$ has the dimension of $s *$

$(p + 1)$ with the hidden neuron values and an additional fictitious hidden neurons with value of 1 for the s training samples. The weight matrix between the hidden layer and the output layer $\boldsymbol{W}^3$ has the dimension of $(p + 1) * m$ with all random values. The dimension of output matrix $\boldsymbol{Y}$ is $s * m$. Each row of $\boldsymbol{Y}$ represents the outputs for one training sample. A matrix $\boldsymbol{D}$ with dimension of $s * m$ is also built to store all outputs data in S training samples as desired outputs data.

The first step is to feedforward the input data to the output neurons. Equation (2.2) can be converted into the matrix format as

$$\boldsymbol{Z}_{s*p} = \sigma(\boldsymbol{X}_{s*(n+1)} \times \boldsymbol{W}^2_{(n+1)*p}) \tag{4.3}$$

where $\boldsymbol{Z}_{s*p}$ represents only the elements in the first p columns are updated and the elements in $(p+1)^{th}$ column of $\boldsymbol{Z}$ remain unchanged. $\sigma(\bullet)$ is the same sigmoid activation function of hidden neurons as Equation (2.2), which applies to each element of the product of $\boldsymbol{X}$ and $\boldsymbol{W}^2$.

To calculate the value of output neurons, Equation (2.2) and Equation (2.7) can be converted into the matrix format as

$$\boldsymbol{Y}_{s*m} = \boldsymbol{Z}_{s*(p+1)} \times \boldsymbol{W}^3_{(p+1)*m} \tag{4.4}$$

where the i^{th} row of $\boldsymbol{Y}$ is the outputs corresponding to the inputs if the i^{th} row of $\boldsymbol{X}$.

The error is calculated by the matrix form of neural network outputs and desired outputs as

$$E = \frac{1}{2} sum((\boldsymbol{Y}_{s*m} - \boldsymbol{D}_{s*m}) \bullet (\boldsymbol{Y}_{s*m} - \boldsymbol{D}_{s*m})) \tag{4.5}$$

where $sum(\bullet)$ is the summation of all the elements in the matrix.

We propose to use the batch mode update method for BP training algorithm as shown in Equation (2.10). The total gradient is the summation of the gradient of all training sample. Let $\boldsymbol{G}^3$ be a $s * m$ matrix representing the gradient at the output layer and $\boldsymbol{G}^2$ be a $s * p$ matrix representing the gradient at the hidden layer. $\boldsymbol{G}^3$ and $\boldsymbol{G}^2$ can be calculated as

$$\boldsymbol{G}^3_{s*m} = \boldsymbol{D}_{s*m} - \boldsymbol{Y}_{s*m} \tag{4.6}$$

$$\boldsymbol{G}^2_{s*p} = \boldsymbol{Z}_{s*p} \bullet (\boldsymbol{U}_{s*p} - \boldsymbol{Z}_{s*p}) \tag{4.7}$$

where $\boldsymbol{U}_{s*p}$ is a $s * p$ matrix with all elements have the value of 1.The weights between the input layer and the hidden layer can be updated as

$$\Delta \boldsymbol{W}^2_{(n+1)*p}\Big|_{W^1=W^1_{now}} = -\eta \bullet \left\{ \boldsymbol{X}^T_{(n+1)*s} \times \left[\boldsymbol{G}^3_{s*m} \times (\boldsymbol{W}^3)^T_{m*p} \bullet \boldsymbol{G}^2_{s*p} \right] \right\} \tag{4.8}$$

$$+\alpha \bullet (\boldsymbol{W}^2_{(n+1)*p}\Big|_{W^1=W^1_{now}} - \boldsymbol{W}^2_{(n+1)*p}\Big|_{W^1=W^1_{old}})$$

And the weights between the hidden layer and the output layer can be updated as

$$\Delta \boldsymbol{W}^3_{(p+1)*m}\Big|_{W^2=W^2_{now}} = -\eta \bullet \left[\boldsymbol{Z}^T_{(p+1)*s} \times \boldsymbol{G}^3_{s*m} \right] \tag{4.9}$$

$$+\alpha \bullet (\boldsymbol{W}^3_{(p+1)*m}\Big|_{W^2=W^2_{now}} - \boldsymbol{W}^3_{(p+1)*m}\Big|_{W^2=W^2_{old}})$$

The CUDA implementation of Equation (4.3) to Equation (4.9) requires both cuBLAS library and CUDA kernel. cuBLAS library is used to implement basic linear algebra operations. It supports three levels of operation: vector operations, matrix-vector operations and matrix-matrix operations. CUDA kernel is the C-like function defined by user that runs on the GPU and performs the same operation on different data. User can easily program any function and CUDA will execute the function on the highly parallel stream processors. Here we illustrate the sigmoid activation function of hidden neurons. Let V be an $s * p$ matrix representing the product of X and W^2 as shown in Equation (4.3). From the knowledge of computer memory architecture, V is stored in a continuous $s * p$ memory space. The operation to V is from the first memory space of V until the last memory space of V.

```
__global__ void Sigmoid(int Quantity, float *Input, float *Output) {
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        if(i < Quantity)
                Output[i] = 1.0/(1.0 + exp(-Input[i]));
        __syncthreads();
}
Sigmoid<<<threadsPerBlock, blocksPerGrid>>>(S * P, V_Dev, Z_Dev);
```

The type of "_global__" for the sigmoid function indicates that the function is running on the GPU. "threadsPerBlock" and "blocksPerGrid" defines the organization of parallel stream processors on the GPU. Each stream processor will have a unique global identifier defined by "blockIdx.x * blockDim.x + threadIdx.x", where "blockIdx.x" is the

block identifier, "blockDim.x" is the block size and "threadIdx.x" is the thread identifier in that block. The first $s * p$ stream processors will execute the sigmoid function and save the result to the corresponding memory space. The "__syncthreads()" routine is used to ensure the synchronization of the stream processors.

The initialization of CUDA program includes allocating memory space and transferring data from system memory to GPU memory. As we propose to include the bias for both weights, additional fictitious input neurons and hidden neurons with values of 1 should be transferred from system memory to GPU memory. Once the matrices are allocated on the GPU memory, they will be kept on the GPU memory with some values updated during training processes. In contrary to the introduced implementation in [50] that the training error is transferred back to the system memory after the feedforward of one training sample, PMAT-G only transfer the total error back to the system memory once per iteration after all the training samples feedforwarded to the output layer. Moreover, the training error is calculated completely on the GPU to take advantage of large number of parallel stream processors on GPU.

Table 4.1 shows the routines of PMAT-G. Step 1 and step 2 are to allocate the GPU memory space and transfer the initialized matrix data from system memory to GPU memory. Step 3 and step 4 are to feedforward input neurons to the output layer. Step 5 calculates the total error on the GPU. Then Weights are updated from step 6 to step 9 with some intermediate matrices to store the computation result. The total error is transferred to the system memory in step 10. CPU will compare the training error with the desired error value in Step 11. If the training error is lower than the desired error, CPU will send an instruction to GPU to transfer the weights to the system memory and

Step	Function	Routine
1	Allocate memory space on GPU	cublasAlloc(X, W, Y, Z, V, D);
2	Transfer matrix data from system memory to GPU Memory	cublasSetMatrix(X, W2, W3, Z, D);
3	Equation (4.3)	cublasSgemm(V, X, W2); Sigmoid(V, Z);
4	Equation (4.4)	cublasSgemm(Y, Z, W3);

5	Equation (4.5)	cublasSrot(Y, D);
		cublasSdot(Y);
		cublasSasum(E, Y);

6	Equation (4.6)	cublasSrot(G3, D, Y);
7	Equation (4.7)	Derivative(G2, Z)

8	Equation (4.8)	cublasSgemm(R, G3, W);
		cublasSrot(R, G2);
		cublasSgemm(W2, X, R);

9	Equation (4.9)	cublasSgemm(W3, Z, G3);
10	Transfer error from GPU memory to system memory	cudaMemcpy(E);
11	If (E < Ed) transfer the weights back to system memory	cublasGetMatrix(W2, W3);

Table 4.1 Routines of Parallel Multiple ANNs Training on Graphics Processing Units (PMAT-G)

CPU will continue to format these weights and write to a file. If the training objective has not been reached, CPU will send an instruction to GPU to repeat step 3 to step 10 until the maximum training iterations has been reached.

More benefits can be obtained from PMAT-G if there are multiple GPUs in the system, i.e., the Hybrid Shared Memory-Graphics Processing Units Architecture (HSMGPUA). Figure 4.4 shows the framework of PMAT-G working on multiple GPUs. If PMAT-G detects multiple GPUs are available in the computer system, it will initialize multiple processors by OpenMP. Each processor is bundled with one GPU by calling "cudaSetDevice()" function provided by CUDA. Let g be the number of GPUs in the system. The total s training samples are distributed to the g GPUs. The input matrix $\boldsymbol{X}$, output matrix $\boldsymbol{Y}$ and the desired output matrix $\boldsymbol{D}$ are scaled to contain nearly s/g rows of data. Each GPU will first execute step 1 to step 5 in Table 4.1 to do initialization, feedforward the input data to the output neurons and calculate training error. Then gradients are calculated on each GPU from step 6 to step 9. However, the weights should not be updated on each GPU independently since the training error and the gradient information on each GPU only contain part of the overall training sample. Data transfers between multiple GPUs have to be performed to ensure one GPU gets information from the other GPUs. The GPU numbered as GPU 1 will collect the training error and the gradient information from all other GPUs and calculate the summation of the training error and the summation of gradients, respectively. The summation of the training error is the total training error of all training samples and it is kept on GPU 1. The summation of the gradient is the total gradient of all training sample and it is transferred to each GPU memory to overwrite the local gradient. Weights then got updated on each GPU's local

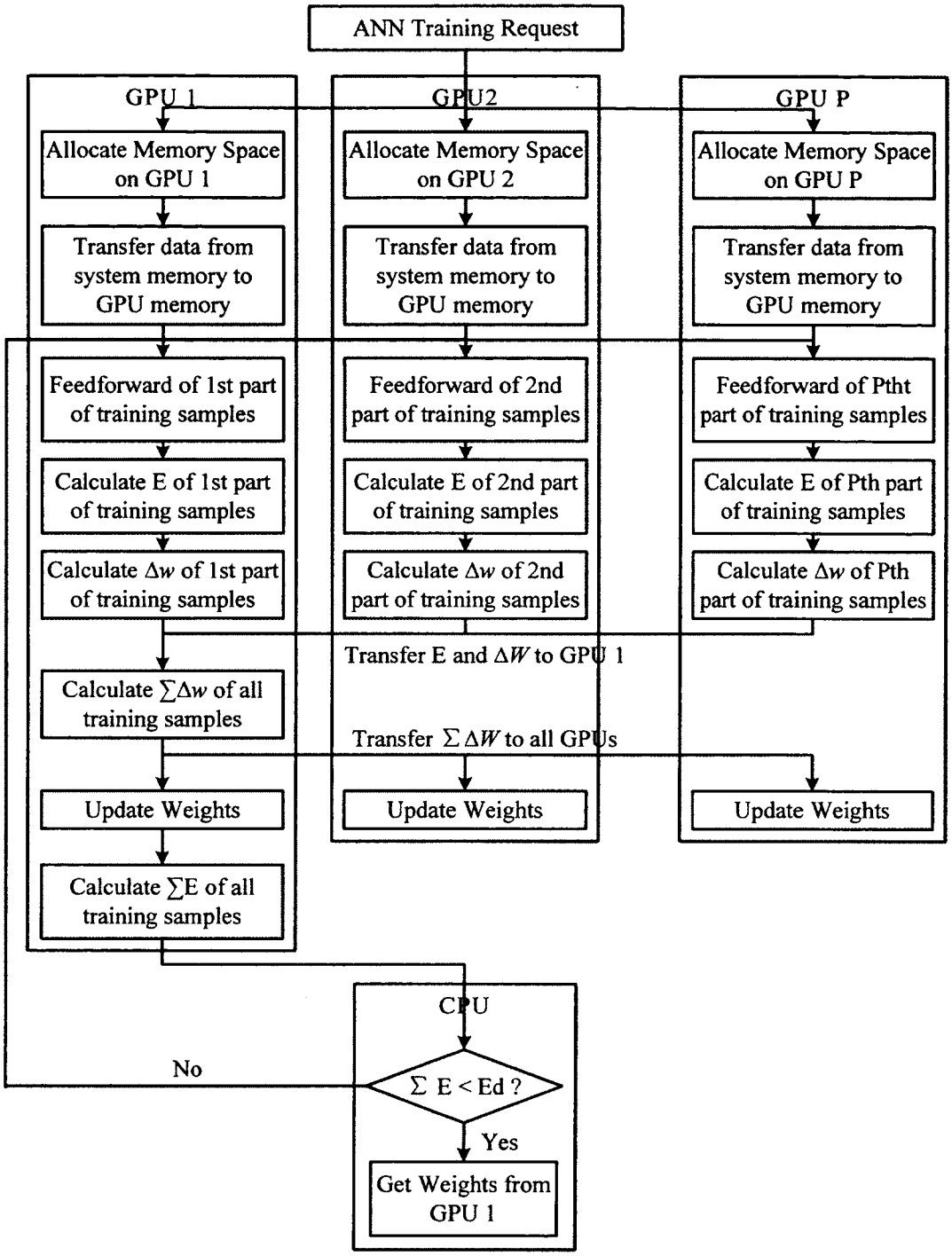


Figure 4.4 Framework of Parallel Multiple ANN Training on multiple GPUs

memory and the values of weights are always the same on all GPU memory spaces. After the weights got updated, CPU will operate alone to send the total training error to CPU while all other GPUs keep waiting. CPU will determine whether the training objective has been reached. If so, CPU will get weights data from GPU 1 and terminate the training process. If not, all GPUs will repeat from step 3 until the maximum training iterations has been reached.

4.4 Verification of Parallel Multiple ANN Training

In order to verify the PMAT-C algorithm and PMAT-G algorithm, we use an iris coupled cavity microwave bandpass filter example as illustrated in Figure 4.5. The filter example is constructed in *CST Microwave Studio*.

As proposed in [74], this cavity filter can be decomposed into seven independent parts. Due to the symmetric structure of this filter, four neural network models can be developed to serve as four modules of the entire MNN. These modules are called sub models. There are two types of ANN structures in this example. Sub model 1~3 have three design variables as inputs, frequency in additional input. Sub model 4 has two design variables as inputs, frequency in additional input. All four sub models have six outputs, which are real and imaginary parts of S_{11}, S_{21} and S_{22}, respectively. Sub model 1~3 have 36072 training samples while sub model 4 has 37408 training samples.

After training data are obtained by simulations from *CST Microwave Studio* for each sub model, the conventional method of sub-models training is to train one sub-model

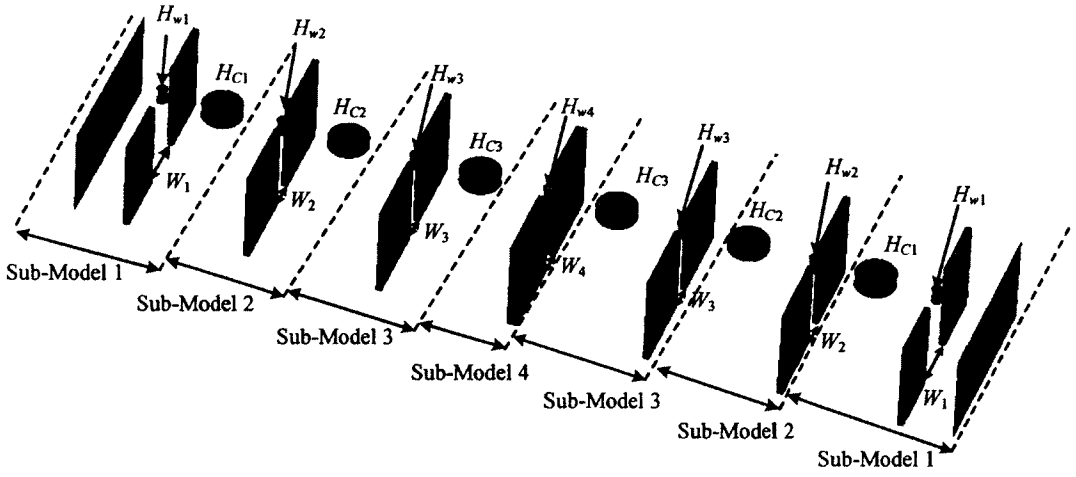


Figure 4.5 Structure of a cavity microwave bandpass filter with structural decomposition

after another. We propose to train these four sub models simultaneously on four computer nodes by PMAT-C algorithm. Each node will be distributed for one sub model and invoke *NeuroModeler Plus* to execute ANN training process. Although structure of sub model 4 is different than the other three sub models, it can still be trained concurrently with the other three sub models since PMAT-C treats one sub model as one independent unit. The processor on each computer node is Intel Xeon E5640.

We also applied PMAT-G into the training of these four sub models. The PMAT-G is running on a system consists of two Nvidia GTX 570 GPUs. Each Nvidia GTX 570 GPU has 480 stream processors formed in highly parallel structures. In order to make the most benefits of multiple GPUs, training samples are distributed on two GPUs. Certain data transfers between the two GPUs are made during the Back Propagation iterations as described in the previous section.

	Training Time					Speed Up
	Sub Model 1	Sub Model 2	Sub Model 3	Sub Model 4	Total	
Conventional Method	13.25 min	12.63	13.59	9.45	48.92	N/A
PMAT-C	13.15	12.97	13.76	9.13	13.86	3.53
PMAT-G	0.26	0.24	0.26	0.17	0.93	52.60

Table 4.2 Speed up of executing Parallel Multiple ANN Training

From Table 4.2, we can see that PMAT-C takes almost the same time as the conventional method to train each ANN. There is a slight difference which is caused by randomly generated weights. However, the conventional method trains ANNs one after another, the total training time is the summation of the training time of each ANN. PMAT-C trains four sub models simultaneously on four computer nodes. The total training time is the longest time of the training time of each ANN. Some overheads of PMAT-C are also included in the total training time, which mainly consists of time for data transfer. PMAT-C achieves a speed up of 3.53 against the conventional method. More speed up is expected to reach if more ANNs are presented.

We can also see that PMAT-G significantly reduced the ANN training time. PMAT-G is 52.60 times faster than the CPU. Due to the larger amount of training data for each

sub model, PMAT-G constructs matrices with large sizes. The matrix and the kernel operations are distributed among the parallel stream processors. PMAT-G is expected to have better performance against CPU with more training samples, since the number of operations increases significantly with the increasing of number of training samples.

4.5 Summary

Parallel Multiple ANN Training on both CPU and GPU has been proposed. PMAT-C distributes multiple ANN training tasks in parallel on multiple computers with multiple processors on each computer. The advantage of PMAT-C is that numerous kinds of training algorithms can be applied to ANN training. Meanwhile, ANNs can be directly trained by matured neural network modeling software, the result files can be directly used for further implementation of neural network, such as the design optimization algorithm based on the neural network.

PMAT-G distributes ANN training on multiple GPUs with multiple stream processors on each GPU. The advantage of PMAT-G is that it achieves very high speed ups by taking advantage of vast number of stream processors on the GPU. With the rapid development of GPU hardware technology, GPU has a brand future for parallel computation.

Chapter 5

Wide-Range Parametric Modeling Technique for Microwave Filters Using Parallel Computational Approach

5.1 Introduction

In recent years, Artificial Neural Networks (ANNs) have been recognized as a powerful technique for parametric microwave device modeling [1]. However, developing an accurate and efficient parametric ANN model with wide-range input parameters is still a challenge. This is because that the number of hidden neurons and amount of data required for ANN training increase very fast with the wider range of input parameters. Conventional ANNs requires a complex structure to learn the input-output relationship. Large amounts of CPU time are consumed on EM simulation for data generation as well as on ANN training. Various advanced techniques have been introduced to reduce the cost of establishing ANN model such as modular neural network [18][74]. However, these techniques are not directly suitable for microwave filters with wide-range parameters.

An efficient Parallel Model Decomposition (PMD) technique for microwave filters with wide-range parameters is proposed. In this technique, the overall ranges of input parameters are decomposed into several small ranges. Multiple ANNs, considered as sub-models, are developed. Training data are then generated within the range of each

sub-model using Design of Experiments (DOE) method to ensure sufficient training data are provided. Sub-models are trained to establish the relationship between input parameters and output responses within their own input parameter ranges. The proposed PMD technique executes training data generation by Parallel Automatic Data Generation (PADG) algorithm. Sub-models are simultaneously trained by integrating Parallel Multiple ANNs Training on Central Processing Unit (PMAT-C) algorithm. A multi-computer with multi-processor environment is applied to achieve maximum speed up. The proposed PMD technique reaches higher efficiency than conventional ANN modeling technique.

5.2 Proposed Parallel Model Decomposition Technique

Developing an accurate parametric ANN model would be challenging for microwave components with wide parameter ranges. Conventional ANN requires large amount of training data due to wide input parameter ranges. It is expensive to generate large amount of data for EM simulation. Meanwhile, the structure of conventional ANN is too complex so that conventional ANN requires large CPU time on ANN training. It is hard to build an accurate and efficient model using a single conventional ANN. We propose to decompose input parameter ranges into smaller parts to develop several independent sub-models. The range of input parameters can be divided into different parts to simplify the structures of the sub-models. Within each sub-model, training data sets are generated within the narrower input parameter ranges. Meanwhile, training data

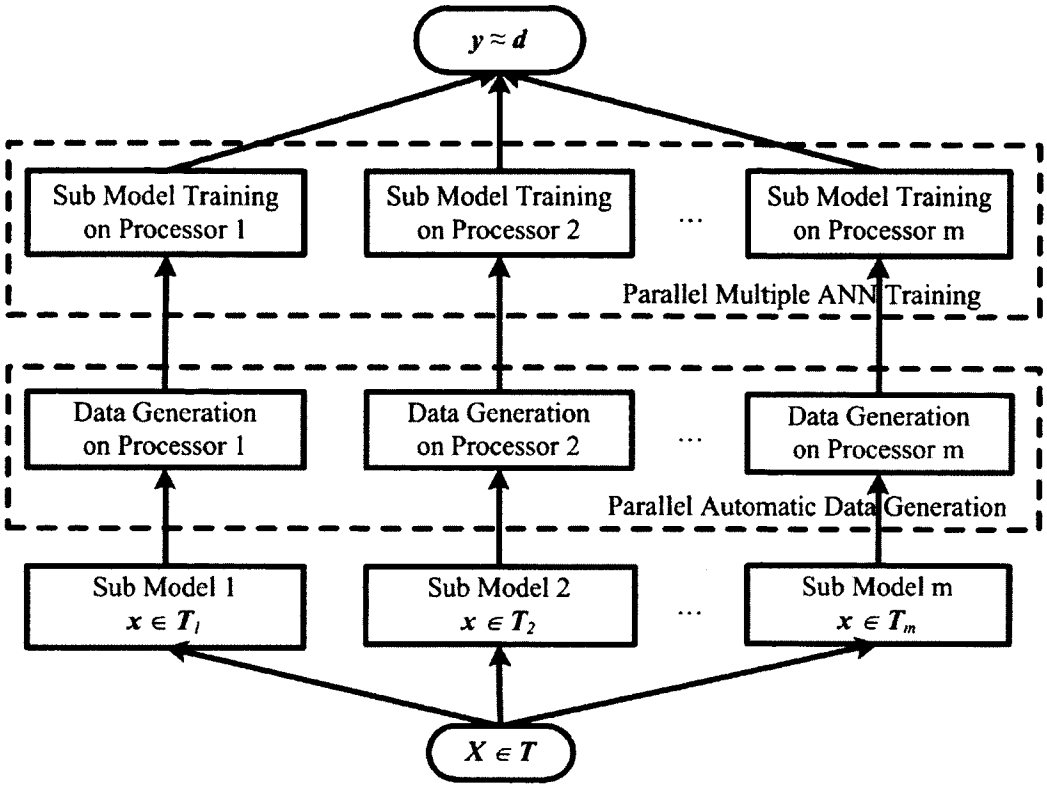


Figure 5.1 Framework of proposed Parallel Model Decomposition (PMD) technique.

generation and sub-model ANN training are executed based on the integration of PADG and PMAT-C to achieve maximum speed up. Since sub-models are developed independently, each sub-model can be trained immediately after its training data has been generated to minimize data transfer. Figure 5.1 shows the framework of proposed PMD technique.

Consider a microwave component with multiple geometrical design parameters, i.e., $X = [X_1, X_2, \ldots, X_n]^T$, where n is the number of input parameters of a model. Given the

maximum and minimum value of j^{th} input parameters as X_j^{min} and X_j^{max}. Let $\boldsymbol{T}$ be an x-space vector of training sets. Let $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, ..., \alpha_n]^T$ be a vector representing the number of divisions of each inputs. The total number of sub-models is $m = \alpha_1 \times \alpha_2 \times ... \times \alpha_n$. Let $\boldsymbol{x}_i$ be a vector representing i^{th} sub-model inputs. The minimum and maximum value of j^{th} input parameters for i^{th} sub-model can be determined as

$$x_{i,j}^{min} = X_j^{min} + \beta_{i,j} \times (X_j^{max} - X_j^{min}) / \alpha_j \tag{5.1}$$

$$x_{i,j}^{max} = x_{i,j}^{min} + (X_j^{max} - X_j^{min}) / \alpha_j \tag{5.2}$$

$$\sum_{j=1}^{n} (\beta_{i,j} \times \prod_{j=0}^{n-1} \alpha_j) = m - 1, \ \alpha_0 = 1 \tag{5.3}$$

Where $\beta_{i,j}$ is the weight of j^{th} input for i^{th} sub-model that can be extracted from Equation (5.3) as

$$\beta_{i,j} = \left\lfloor \left\lfloor \left\lfloor \left\lfloor \left\lfloor (i-1) / \alpha_0 \right\rfloor / \alpha_1 \right\rfloor / \alpha_2 \right\rfloor \cdots \right\rfloor / \alpha_{j-1} \right\rfloor \ mod \ \alpha_j \tag{5.4}$$

where the symbol pair $\lfloor \rfloor$ means the round off number and the operator *mod* represents the remainder number.

After the lower and upper boundaries of the input parameters determined, each sub-model will be trained by an independent ANN. Let $\boldsymbol{x}_i$ be a vector representing input parameters for i^{th} sub-model. Frequency is an additional input. Let $\boldsymbol{y}_i$ be a vector representing model outputs and $\boldsymbol{d}_i$ be a vector representing EM simulations outputs for i^{th} sub-model, respectively. Each sub-model has its own training sets vector $\boldsymbol{T}_i$ so that can be trained individually. The training error of i^{th} sub-model is expressed as

$$E_i = \frac{1}{2}\sum_{x \in T_i}\sum_{f \in F_i}\left|y_i(x, w_i, f) - d_i(x, f)\right|^2$$
$$T_i = \left\{x \mid x_i^{min} < x < x_i^{max}\right\} \tag{5.5}$$

where P is the total number of training geometry samples, F_i is the frequency range of i^{th} sub-model, w_i is a vector containing weight parameters of i^{th} sub-model. x_i^{min} and x_i^{max} are the lower and upper boundary of the i^{th} sub-model defined by Equation (5.1) and Equation (5.2). Each sub-model can has individual frequency range called local frequency range, or each sub-model can has the same frequency range as the other sub-models called global frequency range. The average training error of all sub-models is considered as the overall training error, which is expresses as

$$E = \frac{1}{m}\sum_{i=1}^{m} E_i \tag{5.6}$$

The training objective is to minimize the average training error. Because the computation of training error E_i is completely independent of the computation of E_k, where i is not equal to k, the formulation is naturally suitable for parallel training. Once trained, each sub-model provides sub-response for its own input parameter range with its own frequency range if applied. The combination of all sub-models covers the overall ranges of input parameters.

Parallel computational approaches with the integration of PADG and PMAT-C on hybrid distributed-shared memory architecture are implemented to accelerate training data generation and ANN training procedures. The hybrid distributed-shared memory architecture consists of multiple computers with multiple processors on each computer.

Parallel training data generation tasks are distributed to multiple processors across multiple nodes by strategies defined from Equation (3.1) to Equation (3.3). Parallel multiple ANN training tasks are distributed similarly by strategies defined by Equation (4.1) and Equation (4.2). Since both PADG and PMAT-C are designed to achieve maximum speed up, the parallel computational approach for the proposed PMD technique makes maximum utilization of computation resources to achieve maximum speed up.

5.3 Application Example of a Quasi-Elliptic Filter

5.3.1 50 ~ 70GHz with Global Frequency Range for Each Sub-Model

In order to illustrate the validity of the proposed PMD technique, we use a Quasi-Elliptic filter example shown in Figure 5.2. The filter model has four geometrical design parameters, i.e., $X = [S_1, S_2, S_3, L]^T$. The minimum and maximum values of input parameters are $X^{min} = [70, 90, 170, 380]^T$ and $X^{max} = [130, 130, 230, 420]^T$. Frequency is

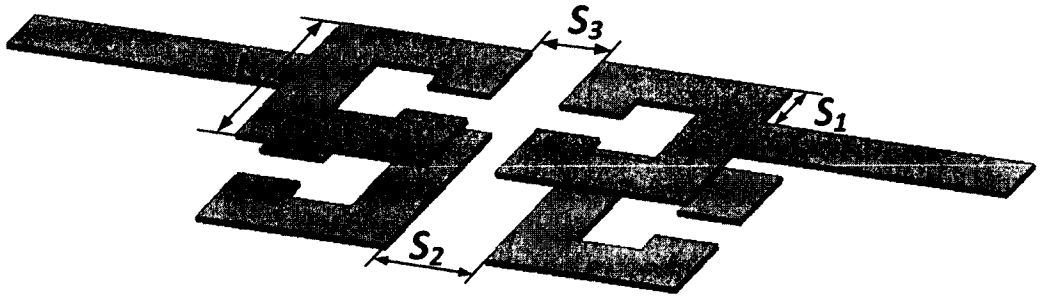


Figure 5.2 Structure of a Quasi-Elliptic filter. This filter model has four wide range geometrical parameters as inputs.

an additional input. Each sub-model has the same frequency range as the other sub-models, which ranges from 50 GHz to 70 GHz with a step size of 0.1 GHz. The model has four outputs, i.e., $y = [RS_{11}, IS_{11}, RS_{21}, IS_{21}]^T$, which are real and imaginary parts of S_{11} and S_{21}, respectively. Four input parameter ranges are decomposed into different parts with division vector $\alpha = [2,2,2,3]^T$, which composes 24 sub-models. The 4-layer perception structure with 15 hidden neurons for each hidden layer is used for each sub-model.

Number of Training Sets	Conventional ANN			Proposed PMD ANN		
	Number of Hidden Neurons per Layer	Training Error (%)	Testing Error (%)	Number of Hidden Neurons per Layer	Training Error (%)	Testing Error (%)
24 * 32 = 768	30	2.13	5.42	15	0.58	1.68
	40	1.85	6.03			
	50	1.53	6.98			
24 * 49 = 1176	30	2.73	4.98	15	0.94	1.26
	40	2.69	5.87			
	50	2.23	6.36			

Table 5.1 Comparison of training results for 24 sub-models with 50 ~ 70 GHz global frequency range

We obtained a small average training error and a small average testing error of for all the 24 sub-models as shown in Table 5.1. Compared with conventional ANN technique, sub-models are trained more accurately and more efficiently after the decomposition of input parameter ranges. Wide-range output responses are too complicated for a conventional neural network to learn the behavior accurately within limited training iterations based on large number of training sets. From Table 5.1, we can see that even with a complex ANN structure, the training error of conventional ANN is still large. Further addition of hidden neurons will make ANN structure more complex and then increase ANN training time, which makes ANN model furthermore less efficiently. The proposed PMD technique develops multiple sub-models with simple structure, and then trains each sub-model independently. If more training data sets are applied, i.e., 49 training data samples for each sub-model and the total number of training data samples for all sub-models is 1176, the average testing error could be even smaller while the average training error is still small. Due to the simpler ANN structure for sub-models with smaller number of hidden neurons than conventional ANN model structure, CPU time of training ANNs sequentially could be reduced as shown in Table 5.3.

Number of Training Sets	24 * 32 = 768	24 * 49 = 1176
Sequential	2334.87(min)	3394.61(min)
Parallel with 24 Threads	112.01(min)	167.27(min)
Speed Up	20.83	20.29

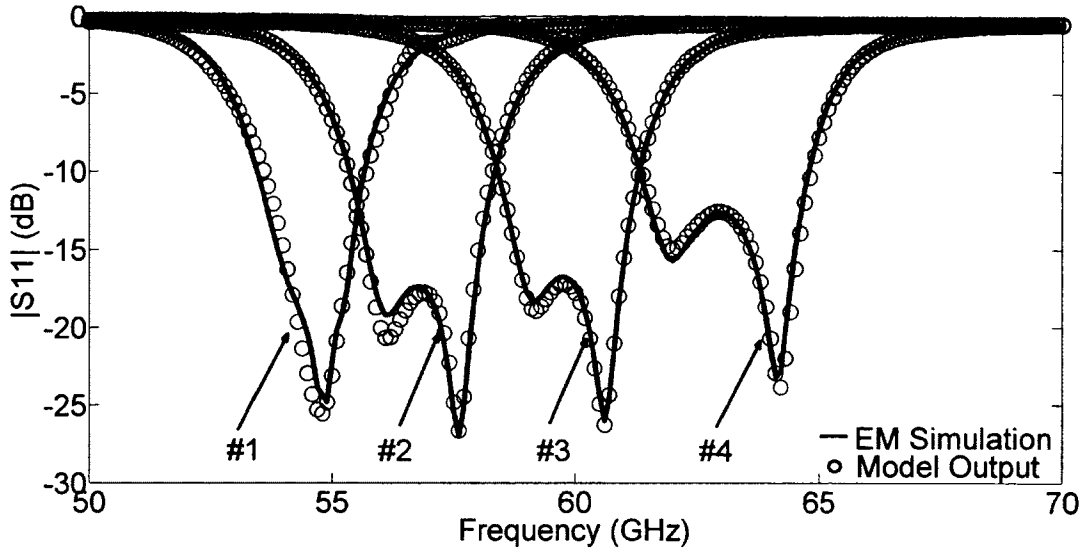
Table 5.2 Comparison of data generation time for 24 sub-models with 50 ~ 70 GHz global frequency range

# of Training Sets	24 * 32 = 768	24 * 49 = 1176
Conventional ANN without Parallel	1091.80(min)	1616.73(min)
Proposed PMD ANN without Parallel	297.43(min)	459.78(min)
Speed Up due to Decomposition	3.67	3.52
Proposed PMD ANN with 24 Parallel Processors	13.54(min)	20.88(min)
Speed Up due to Parallel Computation	21.97	22.02
Total Speed Up by PMD Technique	80.64	77.43

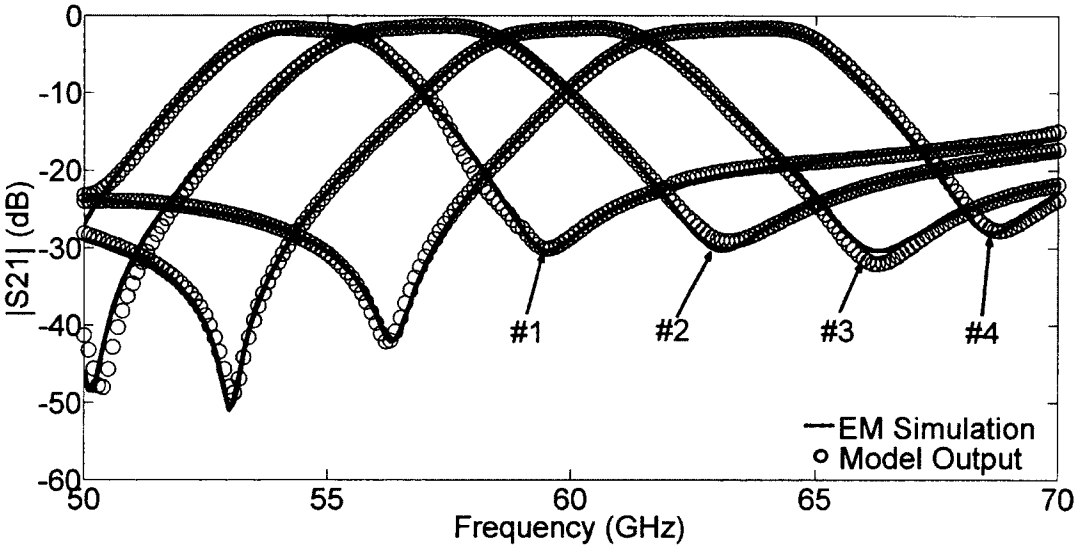
Table 5.3 Comparison of ANN training time for 24 sub-models with 50 ~ 70 GHz global frequency range

Parallel computational approaches are implemented for both EM simulation data generation and ANN training processes on a cluster to achieve high speed ups. The cluster consists of nine computer nodes. Each node is equipped with two quad-core CPUs with hyper-threading technology providing sixteen parallel threads. 24 licenses are available for *Ansoft HFSS* simulator. Parallel task distribution is performed based on Equation (3.1) to Equation (3.3) and Equation (4.1) and Equation (4.2). In this example, the first six nodes use three processors and the last three nodes use two processors, which give a total number of 24 parallel processors for all nine nodes. 768 and 1176 EM simulations could then be done by PADG within 32 and 49 iterations, respectively. Speed ups of over 20 times are achieved as shown in Table 5.2. Furthermore, all 24 sub-models can be trained separately and concurrently on 24 processors across eight nodes. We obtain speed ups of around 22 times for ANN training processes compared with training sub-models sequentially. The parallel computational approach is a powerful method for accelerating both training data generation and ANN model training processes. Meanwhile, it is flexible and expandable when more training data sets and more sub-models are provided.

The accuracy of the proposed PMD technique is confirmed by the comparison of the magnitude of S_{11} and S_{21} of sub-models and EM simulation for four filters as shown in Figure 5.3. The geometrical values and output responses of these four filters covers a wide range. Four filters are modeled by four different sub-models.



(a) Magnitude of S_{11}



(b) Magnitude of S_{21}

Figure 5.3 Comparison of outputs of the proposed ANN model and EM simulation for four filters with parameters belong to four different sub-models.

5.3.2 40 ~ 80GHz with Local Frequency Range for Each Sub-Model

Consider a wider input parameter range for the Quasi-Elliptic filter shown in Figure 5.2. The minimum and maximum values of input parameters are $X^{min} = [40, 120, 150, 320]^T$ and $X^{max} = [140,180, 250, 460]^T$. The frequency range is also expanded to be from 40GHz to 80GHz with a step size of 0.1GHz. Due to the wider input parameter ranges and wider frequency range, we propose to increase the number of divisions for each input parameter. More sub-models are developed and the input parameter ranges in each sub-model are still kept narrow. We use a division vector $\alpha = [3, 3, 3, 4]^T$, which composes 108 sub-models. The minimum and maximum value of each input parameter in each sub-model can be determined by Equation (5.1) to Equation (5.4). In each sub-model, 32 and 49 training data sets are generated as identical to Section 5.3.1 to compare the accuracy of the sub-models, which requires 3456 and 5292 sets of training data to be generated, respectively. ANN training data generation is executed by PADG with 24 parallel threads across nine computer nodes. 3456 and 5292 set of training data can be generated within 144 and 221 iterations, respectively.

Table 5.4 shows the comparison of ANN training data generation time for 108 sub-models by PADG with 24 parallel threads. We obtained speed ups of 20.63 and 20.37 against the conventional sequential training data generation.

# of Training Sets	108 * 32 = 3456	108 * 49 = 5292
Sequential	12900.35(min)	19723.86(min)
PADG with 24 threads	625.32(min)	968.28(min)
Speed Up	20.63	20.37

Table 5.4 Comparison of ANN training data generation time for 108 sub-models

Number of Training Sets	Conventional ANN			Proposed PMD ANN		
	Number of Hidden Neurons per Layer	Training Error	Testing Error	Number of Hidden Neurons per Layer	Training Error	Testing Error
108 * 32 =3456	30	3.59 %	5.82 %	15	1.28 %	2.82 %
	40	2.83 %	6.68 %			
	50	2.25 %	7.33 %			
108 * 49 =5292	30	4.25 %	5.41 %	15	1.56 %	2.33 %
	40	3.94 %	6.27 %			
	50	3.46 %	6.94 %			

Table 5.5 Comparison of ANN training results for 108 sub-models with global frequency range of 40 ~ 80 GHz

ANN training for 108 sub-models is executed by PMAT-C with 108 parallel threads across nine nodes. ANN training task distribution is determined by Equation (4.1) to Equation (4.2) as twelve ANNs are trained simultaneously on each node. From Table 5.5, we can see that small training error and small testing are achieved compared with conventional ANN, while the proposed PMD ANN structure is much simpler than the conventional ANN.

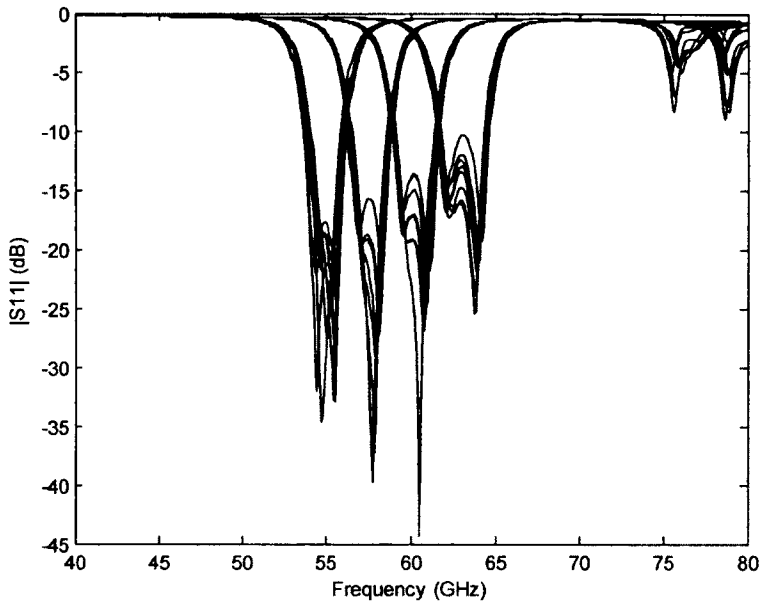
For microwave filters with wide parameters, designers are focused on the specific frequency range where the filters provide best performance. For example, Figure 5.4 shows the magnitude of S_{11} and S_{21} for a set of 32 training data that belong to one sub-model. We can see that these 32 filter structures have the best performance centralized within the frequency range approximately from 49.3GHz to 69.3GHz. This frequency range is actually considered to be the working frequency range for filers with parameter values between the upper and lower boundary of that sub-model.

We propose to apply the frequency range refinement strategy to determine the working frequency range for each sub-model. Each sub-model is proposed to have its own working frequency range for the input parameter value between the minimum and maximum value determined by PMD. The working frequency range is called local frequency range for each sub-model. If frequency range refinement is requested, PMD will search for the frequency points where the first and the last resonances occur for all training sets for each sub-model. The average value of the lowest frequency point and the highest frequency point is referred as the central frequency of a sub-model for the refined

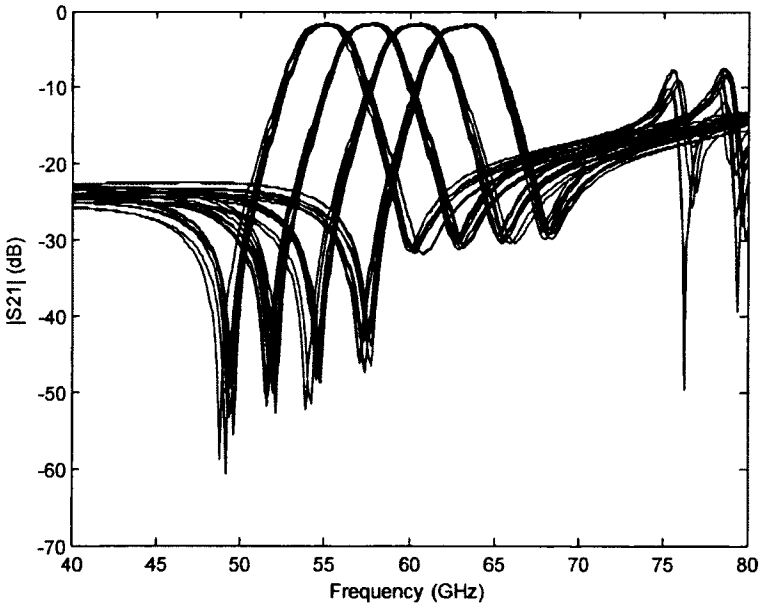
frequency range.

There are two benefits to have local frequency range for each sub-model. There are less frequency points that contained in the training data after applying frequency range refinement. The cut off frequency points are outside the working frequency range so that they are not considered to be modeled by ANNs. Thus, the speed of sub-model ANN training will be significantly increased. Meanwhile, the accuracy of each sub-model will be improved since each sub-model concentrates upon the selected local frequency range. It is easier for sub-models to establish accurate input-output relationship within the individual local frequency range than the global frequency range. The combination of all sub-models with local frequency range still covers the entire global frequency range.

From Figure 5.5, we can see that the local frequency range for this sub-model is refined to be from 49.3 GHz to 69.3 GHz. Compare Figure 5.5 with Figure 5.4, the number of frequency points is reduced from 401 to 201. Each sub-model is applied with the same frequency range refinement strategy to have 201 frequency points within its own individual local frequency range. Each sub-model may has different local frequency range from the other sub-models. The combination of all sub-models covers the entire global frequency range. Table 5.6 shows the comparison of average training error, average testing error and total training time of all 108 sub-models. We can see that the accuracy of sub-models has been increased by applying frequency range refinement strategy. Meanwhile, due to the fewer frequency point, the sub-models training time has been prominently reduced.

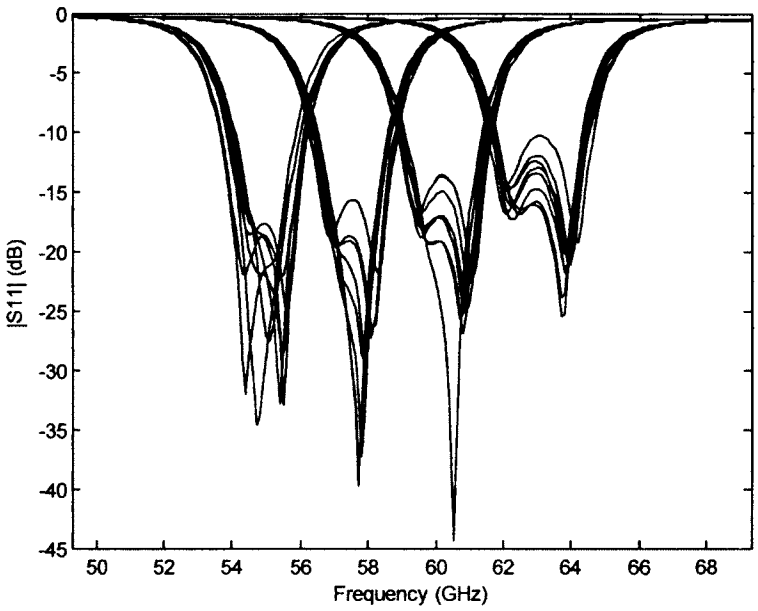


(a) Magnitude of S_{11}

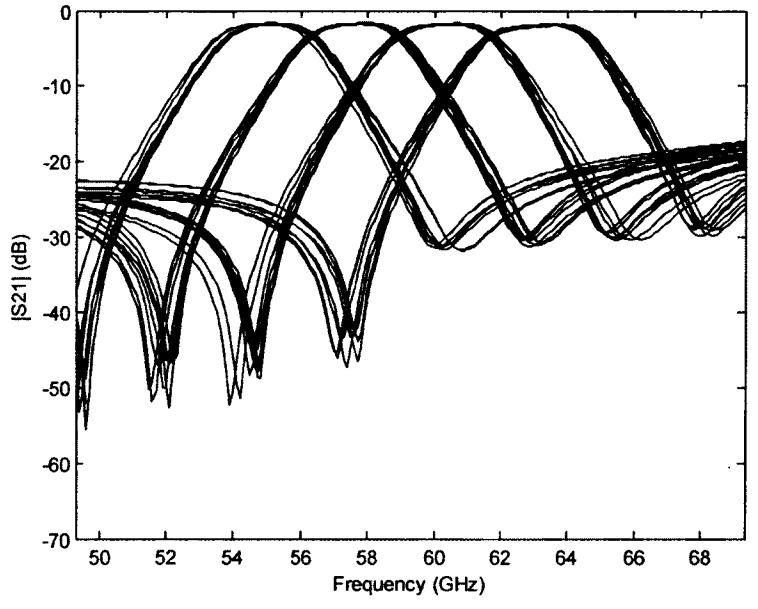


(b) Magnitude of S_{21}

Figure 5.4 Magnitude of S_{11} of 32 training data sets before applying frequency range refinement. The 32 training data sets are generated for one sub-model



(a) Magnitude of S_{11}



(b) Magnitude of S_{21}

Figure 5.5 The magnitude of S_{11} of 32 training data sets after applying frequency selection. The 32 training data sets are generated for one sub-model.

Number of Training Sets	Before Frequency Selection			After Frequency Selection		
	Training Error	Testing Error	Training Time	Training Error	Testing Error	Training Time
108 * 32= 3456	1.28 %	2.82 %	59.99 (min)	0.81 %	2.08 %	29.76 (min)
108 * 49 = 5292	1.56 %	2.33 %	93.18 (min)	1.18 %	1.74 %	45.68 (min)

Table 5.6 Comparison of proposed PMD ANN training results and training time by parallel multiple ANN training on CPU before and after frequency selection

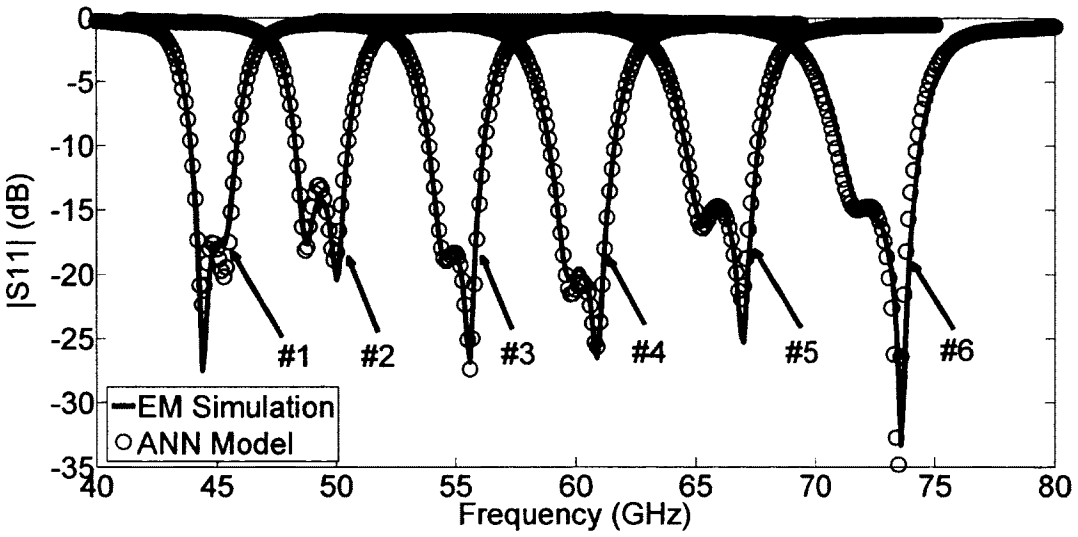
# of Training Sets	108 * 32 = 3456	108 * 49 = 5292
Conventional ANN without Parallel	6285.66 (min)	9482.08 (min)
Proposed PMD ANN without Parallel	1681.14 (min)	2614.27 (min)
Speed Up due to Decomposition	3.74	3.63
Proposed PMD ANN with 108 Parallel Processors	29.76 (min)	45.68 (min)
Speed Up due to Parallel Computation	56.49	57.23
Total Speed Up by PMD Technique	211.21	207.74

Table 5.7 Comparison of ANN training time for 108 sub-models with local frequency ranges

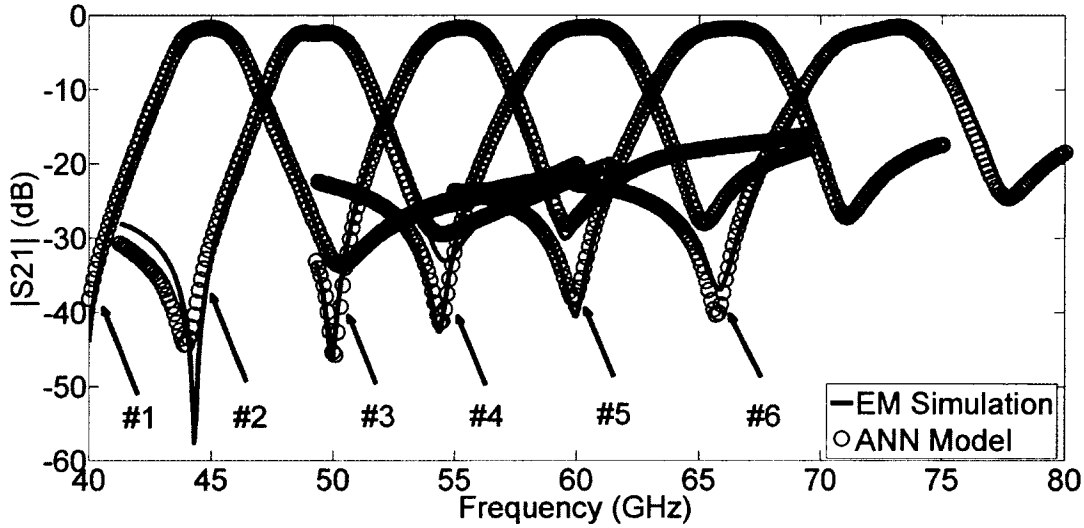
The comparison of sub-models training time with conventional ANN is shown in Table 5.7. We can see that due to the sample structure and local frequency range of each sub model, speed ups of 3.74 and 3.63 have been achieved by the model decomposition technique for 32 and 49 training sets for each sub-model, respectively. The 108 sub-models training is executed simultaneously on 108 parallel threads, which lead to speed ups of 56.49 and 57.23 due to parallel multiple ANN training. PMD technique achieves over 200 speed ups against conventional ANN. PMD technique provides an accurate and efficient modeling technique with frequency selection strategy. Figure 5.6 shows the comparison of the magnitude of S_{11} and S_{21} of sub-models and EM simulation for six filters, which are modeled by six different sub-models.

5.4 Summary

An efficient parametric modeling technique for microwave components with wide-range parameters has been propose. The proposed Parallel Model Decomposition (PMD) technique decomposes the range of input parameters of ANN model into several smaller ranges to develop multiple sub-models. A unified algorithm has been proposed to determine the lower and upper boundaries of input parameters for each sub-model. The proposed PMD technique executes training data generation by the PADG technique and multiple sub-models training by PMAT-C technique. The proposed sub-modeling technique provides an accurate and fast prediction of the EM behavior of microwave components with wide-range parameters.



(a) Magnitude of S_{11}



(b) Magnitude of S_{21}

Figure 5.6 Comparison of outputs of the proposed ANN model and EM simulation for six filters with parameters belong to six different sub-models.

Chapter 6

Conclusions and Future Research

6.1 Conclusions

This thesis has presented wide-range parametric modeling technique for microwave components utilizing parallel computational approaches as Parallel Automatic Data Generation (PADG) technique and Parallel Multiple ANN Training (PMAT) technique.

Data generation is the most computationally intensive stage in Artificial Neural Networks (ANNs) modeling technique since the detailed EM/physics/circuit simulation are usually CPU expensive. The proposed PADG technique accelerates data generation process on hybrid distributed-shared memory computer architecture and achieves maximum speed up based on the available computational resources. The PADG technique distributes simulation tasks to multiple processors across multiple interconnect computers and drives multiple simulators simultaneously on all parallel processors. A unified task distribution strategy has been proposed to automatically distribute simulation tasks based on the computational resources provided by user. Application example of data generation by *Ansoft HFSS* for an interdigital band-pass filter has demonstrated that the proposed PADG technique achieves high speed up and high parallel efficiency on a

cluster. Therefore, the PADG technique can be used in any neural network data generation stage.

We have also introduced two parallel approaches for multiple ANNs training. During the ANN training stage, multiple ANNs can be trained concurrently on multiple parallel processors by the proposed Parallel Multiple ANN Training on Parallel Multiple ANNs Training on Central Processing Unit (PMAT-C) technique. The ANN training on each processor can be executed either by classic ANN training algorithms or invoking neural network modeling software. ANN training task distribution strategy has been presented. Another novel parallel approach is to parallelize the Back Propagation (BP) algorithm on a computer with multiple Graphics Processing Units (GPUs). The proposed Parallel Multiple ANNs Training on Graphics Processing Unit (PMAT-G) technique uses the batch mode update method of BP algorithm and takes full advantages of the highly parallel structure of GPUs. The implementation of BP algorithm on multiple GPUs has been proposed for the first time with data transfer between multiple GPUs. The proposed PMAT-G technique minimizes the overhead of data transfer between GPU memory and system memory. A modular neural network application example has been presented to demonstrate the advantages of both PMAT-C technique and PMAT-G technique in multiple ANNs training.

An efficient parametric modeling technique for microwave components with wide-range parameters has been introduced. The proposed Parallel Model Decomposition (PMD) technique decomposes the range of input parameters of ANN model into several smaller ranges to develop multiple sub-models. Each input parameter range can be decomposed to different parts than the others. A unified algorithm has been proposed to

determine the lower and upper boundaries of input parameters for each sub-model. The data generation stage is executed by PADG technique. The sub-models training are implemented by PMAT-C technique. Compared with conventional ANN modeling technique, the proposed PMD technique achieves higher accuracy and higher efficiency with simple sub-model structure by using parallel computational approaches. The proposed sub-modeling technique provides an accurate and fast prediction of the EM behavior of microwave components with wide-range parameters.

6.2 Future Research

Artificial Neural Networks have been proved as a powerful technology for RF and microwave modeling and design. Neural networks are genetic that they can be achieved at all aspects of RF/microwave design such as modeling, simulation, optimization and synthesis. The development of the PADG technique has addressed the acceleration of data generation by microwave simulators. As neural networks can be utilized at all levels of RF/microwave design including circuits and systems, the next step of the PADG development is to provide a universal module that can be easily programmed by user to drive any kind of existing and future simulators.

As demonstrated in the thesis, GPUs can provide remarkable performance gains than CPUs for intensive computations. An interesting topic following the idea of the PMAT-G technique would be the development of BP algorithm for multiple hidden layers. Other ANN training algorithms also have potential to be parallelized on GPUs to achieve conspicuous speed ups. Moreover, the PMAT-G technique can also be integrated into the

Parallel Automatic Model Generation (PAMG) algorithm to further reduce the cost of developing an accurate ANN model.

Based on the benefits of the PMD technique, another interesting topic would be developing an automatic modeling decomposition algorithm for any ANN model with complex structure. The algorithm should accurately predict the complexity of ANN model structure and decompose the input parameter ranges of ANN model. It is also desirable to automatically deicide how many sub-models to be developed.

Expanding the proposed techniques in this thesis with these suggestions would make neural network model development more efficient and more intelligent. These techniques would further enable the RF/microwave designers to obtain benefit of apply neural network technology.

Bibliography

[1] Q. J. Zhang and K. C. Gupta, *Neural Networks for RF and Microwave Design*, Norwood, MA: Artech House, 2000.

[2] P. M. Watson and K. C. Gupta, "Design and optimization of CPW circuits using EM-ANN models for CPW components," *IEEE Trans. Microwave Theory Tech.*, vol. 45, no. 12, pp. 2515–2523, Dec. 1997.

[3] B. Davis, C. White, M. A. Reece, M. E. Bayne, Jr., W. L. Thompson, II, N. L. Richardson, and L. Walker, Jr., "Dynamically configurable pHEMT model using neural networks for CAD," *IEEE MTT-S Int. Microw. Symp. Dig.*, Philadelphia, PA, Jun. 2003, vol. 1, pp. 177–180.

[4] J. P. Garcia, F. Q. Pereira, D. C. Rebenaque, J. L. G. Tornero, and A. A. Melcon, "A neural-network method for the analysis of multilayered shielded microwave circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 54, no. 1, pp. 309–320, Jan. 2006.

[5] M. Isaksson, D. Wisell, and D. Ronnow, "Wide-band dynamic modeling of power amplifiers using radial-basis function neural networks," *IEEE Trans. Microwave Theory Tech.*, vol. 53, no. 11, pp. 3422–3428, Nov. 2005.

[6] H. Kabir, Y. Wang, M. Yu, and Q. J. Zhang, "Neural network inverse modeling and applications to microwave filter design," *IEEE Trans. Microwave Theory Tech.*, vol. 56, no. 4, pp. 867–879, Apr. 2008.

[7] V. K. Devabhaktuni, M. C. E. Yagoub, and Q. J. Zhang, "A robust algorithm for automatic development of neural-network models for microwave applications," *IEEE Trans. Microwave Theory Tech.*, vol. 49, no. 12, pp. 2282–2291, Dec. 2001.

[8] R. Biernacki, J. W. Bandler, J. Song, and Q. J. Zhang, "Efficient quadratic approximation for statistical design," *IEEE Trans. Circuit Syst.*, vol. CAS-36, pp. 1449-1454, 1989.

[9] P. Meijer, "Fast and smooth highly nonlinear multidimensional tale models for device modeling," *IEEE Trans. Circuit Syst.*, vol. 37, pp. 335-346, 1990.

[10] A. H. Zaabab, Q. J. Zhang and M. S. Nakhla, "A neural network approach to circuit optimization and statistical design," *IEEE Trans. Microwave Theory Tech.*, vol. 43, pp. 1349-1358, 1995.

[11] Q. J. Zhang, F. Wang and M. S. Nakhla, "Optimization of high-speed VLSI interconnects: A review," *Int. J. of Microwave and Millimeter-Wave CAE*, vol. 7, pp. 83-107, 1997.

[12] F. Wang and Q. J. Zhang, "Knowledge-based neural models for microwave design," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 2333-2343, 1997.

[13] J. E. Rayas-Sanchez, "EM-based optimization of microwave circuits using artificial neural networks: The state-of-the-art," *IEEE Trans. Microwave Theory Tech.*, vol. 52, no. 1, pp. 420–435, Jan. 2004.

[14] S. Koziel and J. W. Bandler, "A spacemapping approach to microwave device modeling exploiting fuzzy systems," *IEEE Trans. Microwave Theory Tech.*, vol. 55, no. 12, pp. 2539–2547, Dec. 2007.

[15] L. Zhang, Y. Cao, S. Wan, H. Kabir and Q. J. Zhang, "Parallel Automatic Model Generation Technique for Microwave Modeling," *IEEE MTT-S,* Honolulu, HI, pp. 103 - 106, Jun. 2007.

[16] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th Edition, San Francisco, CA: Morgan Kaufmann, 2011

[17] Z. Lu, X. An, and W. Hong, "A fast domain decomposition method for solving three-dimensional large-scale electromagnetic problems," *IEEE Trans. Antennas Propagat.*, vol. 56, no. 8, pp. 2200–2210, Aug. 2008.

[18] H. Kabir, Y. Wang, M. Yu and Q. J. Zhang, "High-Dimensional Neural-Network Technique and Applications to Microwave Filter Modeling," *IEEE Trans. Microwave Theory Tech.*, vol. 58, no. 1, pp. 145-156, Jan. 2010.

[19] Q. J. Zhang, M. C. E. Yagoub, X. Ding, D. Gouletter, R. Sheffield and H. Feyzbakhsh, "Fast and accurate modeling of embedded passives in multi-layer printed circuits using neural network approach," *Elect. Components & Tech. Conf.*, pp. 700-703, San Diego, CA, 2002.

[20] K. C. Gupta, "Emerging trends in millimeter-wave CAD," *IEEE Trans. Microwave Theory and Techniques*, vol. MTT-46, pp. 747-755, June, 1998.

[21] T. Itoh, *Numerical Techniques for Microwave and Millimeter-Wave Passive Structures*, New York: John Wiley and Sons, 1989.

[22] M. N. O. Sadiku, *Numerical Techniques in Electromagnetics*, 2nd Edition, Boca Raton, FL: CRC Press, 2000.

[23] A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 3rd Edition, Norwood, MA: Artech House, 2005.

[24] P. M. Waston and K. C. Gupta, "EM-ANN models for microstrip vias and interconnects in dataset circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 44, pp. 2495-2503, 1996.

[25] G. L. Creech, B. Paul, C. Lesniak and M. Calcatera, "Artificial neural networks for accurate microwave CAD application," *IEEE Int. Microwave Symp.*, pp. 733-736, San Francisco, CA, 1996.

[26] V. B. Litovski, J. Radjenovic, Z. M. Mrcarica and S. L. Milenkovic, "MOS transistor modeling using neural network," *Electronics Lett.*, vol. 28, pp. 1766-1768, 1992.

[27] V. K. Devabhaktuni, C. Xi and Q. J. Zhang, "A neural network approach to the modeling of heterjunction bipolar transistors from S-parameter data," *Euro. Microwave Conf.*, pp. 306-311, Amsterdam, Netherlands, 1998.

[28] K. Shirakawa, M. Shimizu, N. Okubo and Y. Daido, "Structural determination of multilayered large signal neural network HEMT model," *IEEE Trans. Microwave Theory Tech.*, vol. 46, pp. 1367-1375, 1998.

[29] Y. H. Fang, M. C. E. Yagoub, F. Wang and Q. J. Zhang, "A new macromodeling approach for nonlinear microwave circuits based on recurrent neural networks," *IEEE Trans. Microwave Theory Tech.*, vol. 48, pp. 2335-2344, 2000.

[30] X. Ding, J. J. Xu, M. C. E Yagoub and Q. J. Zhang, "A combined state space formulation/equivalent circuit and neural network technique for modeling of embedded passives in multilayer printed circuits," *Applied Computational Electromagnetics Society Journal*, vol. 18, no. 2, pp. 89-97, 2003.

[31] L. Ton, Y. Cao, J. J. Xu and Q. J. Zhang, "Recent advances in neural based time domain EM modeling and simulation," *10th Int. Symp. on Antenna Technology and Applied Electromagnetics 2004/URSI*, Ottawa, Canada, July, 2004.

[32] L. Ton, J. J. Xu, Q. J. Zhang, R. Sheffield, H. Kwong and L. Marcanti, "Modeling of embedded passives in multi-layer printed circuits using neural networks", *Int. Conf. on Embedded Passives.*, San Jose, California, June, 2004.

[33] A. Veluswami, M. S. Nakhla and Q. J. Zhang, "The application of neural network to EM-based simulation and optimization of interconnects in high speed VLSI circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 45, pp. 712-723, 1997.

[34] G. Kothapali, "Artificial neural network as aids in circuit design," *Microelectronics Journal*, vol. 26, pp. 598-678, 1995.

[35] Q. J. Zhang and M. S. Nakhla, "Signal integrity analysis and optimization of VLSI interconnects using neural network models," *IEEE Int. Symp. Circuits Systems.*, pp. 459-462, London, England, 1994.

[36] T. Hong, C. Wang and N. G. Alexopoulos, "Microstrip circuit design using neural network", *IEEE Int. Symp.Dig*, pp. 413-416, Atlanta, Georgia, 1993.

[37] M. D. Baker, C. D. Himmel and G. S. May, "In-situ prediction of reactive ion etch endpoint using neural network," *IEEE Trans. Components, Packaging, and Manufacturing Tech. Part A.*, vol. 18, pp. 478-483, 1995.

[38] M. Vai, S. Wu, B. Li and S. Prasad, "Reverse modeling of microwave circuits with bidirectional neural network models," *IEEE Trans. Microwave Theory Tech.*, vol. 46, pp. 1492-1494, 1998.

[39] J. W. Bandler, M. A. Ismail, J. E. Rayas-Sánchez and Q. J. Zhang, "Neuromodeling of microwave circuits exploiting space mapping technology," *IEEE Trans. Microwave Theory Tech.*, vol. 47, pp. 2417-2427, 1999.

[40] S. Goasguen, S. M. Hammadi and S. M. El-Ghazaly, "A global modeling approach using artificial neural network," *IEEE MTT-S Int. Microwave Symp. Digest*, pp. 153-156, Anaheim, CA, 1999.

[41] M. Vai and S. Prasad, "Automatic impedance matching with a neural network," *IEEE Microwave Guided Wave Letter*, vol. 3, pp, 353-354, 1993.

[42] L. Zhang, Y. Cao, S. Wan, H. Kabir and Q. J. Zhang, "Parallel Automatic Model Generation Technique for Microwave Modeling", *IEEE Int. Microwave Symp.*, pp. 103 - 106, Honolulu, HI, Jun. 2007.

[43] D. E. Rubelhart, G. E. Hinton and R. J. Williams, "Learning internal presentations by error propagation," in *Parallel Distributed Processing*, vol. 1, D. E. Rumelhart and J. L. McClelland, Eds., Cambridge, MA: MIT Press, pp. 318-362, 1986.

[44] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd Edition, Upper Saddle River, HJ: Prentice Hall, 1999.

[45] H. Robbins and S. Monoro, "A stochastic approximation method," *Annual of Mathematical Statistics*, vol. 22, pp. 400-407, 1951.

[46] C. Darken, J. Chang and J. Moody, "Learning rate schedule for faster stochastic gradient search," In *Neural Network for Signal Processing*, vol. 2, S. Y. Kung, F. Fallside, J. A. Sorensen and C. A. Kamm, Eds., IEEE Workshop, IEEE Press, pp. 3-13, 1992

[47] *Neural Network Toolbox: For Use with Matlab*, The MathWorks Inc., Natick, MA, 1993

[48] A. A. Marcelo, P. T. Edilberto, R. C. Fabio and P.A. Joao, "Parallel training for neural networks using PVM with shared memory," *Proc. on Evolutionary Computation*, Canberra, Australia, pp. 1315-1322, Dec. 2003.

[49] RK. Thulairam, RM. Rahman, and P. Thulasiraman, "Neural network training algorithms on parallel architectures for finance applications," *Proc. Intl. Conf on Parallel Processing Workshops*, Kaohsiung, Taiwan, pp. 236 – 243, Oct. 2003.

[50] X. Sierra-Canto, F. Madera-Ramírez and V. Uc-Cetina, "Parallel training of a back-propagation neural network using CUDA," *2010 Ninth Int. Conf. on Machine Learning and Applications*, Washington, pp. 307-312, Dec. 2010.

[51] S. Scanzio, S. Cumani, R. Gemello, F. Mana and P. Laface, "Parallel implementation of artificial neural network training," *2010 IEEE Int. Conf. on Acoustics Speech and Signal Processing*, Dallas, TE, pp. 4902-4905, Mar. 2010.

[52] H. El-Rewini, M. Abd-El-Barr, *Advanced Computer Architecture and Parallel Processing*, Hoboken, NJ: John Wiley & Sons, 2005

[53] *MPI: A Message-Passing Interface Standard,* Version. 2.2, Message Passing Interface Forum, Sep. 2009.

[54] *OpenMP Application Interface,* Version. 3.1, OpenMP Architecture Review Board, Jul. 2011.

[55] P. Pacheco, *An Introduction to Parallel Programming*, San Francisco, CA: Morgan Kaufmann, 2011

[56] D. Steinkraus, I. Buck, P. Y. Simard, "Using GPUs for machine learning algorithms," *Proceedings of the 2005 Eighth Int. Conf. on Document Analysis and Recognition*, vol. 2, pp. 1115-1120, Seoul, Korea, 2005

[57] E. W. Lowe, M. Butkiewicz, N. Woetzel, J. Meiler, "GPU-accelerated machine learning techniques enable QSAR modeling of large HTS data," *2012 IEEE Symp. on Computational Intelligence in Bioinformatics and Computational Biology*, pp. 314-320, San Diego, CA, May. 2012.

[58] *Parallel Computing Toolbox User's Guide*, Version 2012a, The MathWorks Inc., Natick, MA, 2012.

[59] R. Abdalkhalek, O. Coulaud, G. Latu, "Fast seismic modeling and Reverse Time Migration on a GPU cluster," *2009 Int. Conf. on High Performance Computing & Simulation*, pp. 36-43, Leipzig, Germany, Jun. 2009.

[60] *The OpenCL Specification*, Version. 1.2, Document Revision. 1.5, Khronos Group, Beaverton, Oregon, Nov. 2011

[61] *NVIDIA CUDA C Programming Guide*, Version. 4.2, Nvidia Corporation, Santa Clara, CA, May. 2012

[62] *HFSS*, Version 13.0, Ansys Inc., Pittsburgh, PA, 2010

[63] *Advanced Design System*, Version 2011.10, Agilent Technologies, Santa Clara, CA, 2011

[64] *CST Microwave Studio*, Version 2011, Computer Simulation Technology, Darmstadt, Germany, 2011

[65] Q. J. Zhang, L. Ton and Y. Cao, "Microwave Modeling Using Artificial Neural Networks and Applications to Embedded Passive Modeling," *2007 Int. Conf. on Microwave and Millimeter Wave Technology*, pp. 1-4, Guilin, China, Apr. 2007.

[66] V. Gongal Reddy, S. Zhang, Y. Cao, Q. J. Zhang, "Efficient Design Optimization of Microwave Circuits using Parallel Computational Methods", *European Microwave Week*, Amsterdam, Netherlands, Oct. 2012 (Accepted, to be published)

[67] U. Lahiri, A. K. Pradhan, and S. Mukhopadhyaya, "Modular neural network-based directional relay for transmission line protection," *IEEE Trans. Power Syst.*, vol. 20, no. 4, pp. 2154–2155, Nov. 2005.

[68] Z. Lu, X. An, and W. Hong, "A fast domain decomposition method for solving three-dimensional large-scale electromagnetic problems," *IEEE Trans. Antennas Propagat.*, vol. 56, no. 8, pp. 2200–2210, Aug. 2008.

[69] *NeuroModeler Plus*, Version 2.1E, Q. J. Zhang, Carleton University, Ottawa, Canada, 2008.

[70] *Linear Algebra PACKage Users' Guide,* 3rd Edition, Society for Industrial and Applied Mathematics, Philadelphia, PA, Aug. 1999.

[71] *OpenCL Basic Linear Algebra Subprograms*, Advanced Micro Devices, Inc., Sunnyvale, CA, Dec. 2011.

[72] *Intel Math Kernel Library*, Intel Corporation, Santa Clara, CA, Aug. 2011.

[73] *CUDA Toolkit 4.2 CUBLAS Library*, Nvidia Corporation, Santa Clara, CA, Feb. 2012

[74] Y. Cao, S. Reitzinger and Q. J. Zhang, "Simple and Efficient High-Dimensional Parametric Modeling for Microwave Cavity Filters Using Modular Neural

Network", *IEEE Microw. Wireless Compon. Lett.*, vol. 21, no. 5, pp. 258-260, May. 2011.