POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Master of Science in

Computer Engineering

NEURAL NETWORK MODELING OF A CUSTOMIZABLE OPENCL STEREO-MATCHING FOR A MULTI-CLUSTER, INDUSTRIAL ARCHITECTURE

Supervisor: Prof. Vittorio Zaccaria

Co-Supervisor: Prof. Cristina Silvano

Master Graduation Thesis by: Nazanin Vahabi

Student Id. number: 750234

Academic Year: 2011/2012

POLITECNICO DI MILANO

Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

Corso di Laurea Specialistica in

Ingegneria Informatica

Modellazione basata su reti neurali di un'applicazione di stereo-matching sviluppata in OpenCL per un'architettura multi-cluster industriale

Relatore: Prof. Vittorio Zaccaria

Correlatore: Prof. Cristina Silvano

Tesi di laurea di: Nazanin Vahabi

matr. 750234

Anno Accademico 2011/2012

NEURAL NETWORK MODELING OF A CUSTOMIZABLE OPENCL STEREO-MATCHING FOR A MULTI-CLUSTER, INDUSTRIAL ARCHITECTURE

A Thesis Project

Submitted to the Department of

Computer Engineering

at Politecnico di Milano

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science in Computer Engineering

Adviser: Vittorio Zaccaria, Ph.D.

Co-Adviser: Cristina Silvano, Ph.D.

Nazanin Vahabi

JULY 2012

ABSTRACT IN ENGLISH

In recent years, application-specific multiprocessor systems-on-chip (MPSoCs) have become more complex. Generally, these architectures are designed by using a platform-based approach. Finding the best trade-off in terms of the selected figures of merit (such as energy, delay, and area) can be achieved by tuning a wide range of customizable parameters. This optimization phase is called Design Space Exploration (DSE), and it usually consists of a multi objective optimization problem with multiple constraints. So far, several heuristic techniques have been proposed to address the DSE problem for MPSoC, but they were not efficient enough in order to identify the Pareto front of feasible solutions in a reasonable amount of time and manage the application-specific constraints.

The methodology proposed in this dissertation is an efficient DSE method for application-specific MPSoC. This methodology combines the design of experiments (DoEs) and response surface modeling (RSM) techniques for managing system-level constraints. The main target is to find the optimal parameterized configurations of either architectures and/or applications by using the minimum number of simulations.

First, the DoE phase generates an initial plan of experiments used to create a coarse grain view of the target design space to be explored by simulations. Then, Artificial Neural Network (ANN) is used to refine the simulation-based exploration. The RSM-based techniques provide an analytical representation of the configurations. ANNs tackle the problem of DSE by reducing the time required to evaluate a system configuration. To trade-off the efficiency in terms of time and accuracy of the proposed technique, a set of experimental results for the customization of a multi-cluster application has been reported in this dissertation.

Keywords: Application-specific Processors, Design Space Exploration, Chip Multi-processors, Artificial Neural Network

ABSTRACT IN ITALIAN

Negli ultimi anni, i sistemi multi-processore on-chip sono diventati sempre più complessi. In generale queste architetture sono progettate usando un approccio basato su piattaforma, ove si raggiunge un trade-off ottimale in termini di energia, ritardo e area, modificando un largo insieme di parametri.

Questa fase di ottimizzazione è chiamata Design Space Exploration e tipicamente consiste nella risoluzione di un problema multi-obiettivo con più vincoli.

Fino ad ora sono state proposte diverse euristiche per indirizzare il problema della DSE per i sistemi multi-processore; tali euristiche pero' non sono state efficienti abbastanza per identificare i fronti di Pareto del problema in tempo ragionevole e rispettando i vincoli specifici dell'applicazione.

La metodologia proposta in questa tesi consente di effettuare la Design Space Exploration in maniera efficiente. La metodologia combina la progettazione degli esperimenti (DoE) e la modellazione basata su superfici di risposta per gestire l'ottimizzazione a livello di sistema. L'obiettivo principale è quello di trovare configurazioni ottimali sia architetturali che applicative con un basso numero di simulazioni.

La metodologia utilizza dapprima una progettazione degli esperimenti per ottenere una vista a grana grossa dello spazio di esplorazione. In seguito, le reti neurali sono utilizzate per raffinare l'esplorazione basata su simulazioni. Le tecniche basate su superfici di risposta come le reti neurali offrono una rappresentazione analitica delle configurazioni e permettono di indirizzare il problema della Design Space Exploration riducendo tempo dedicato a valutare le varie configurazioni. Per analizzare la tecnica proposta sia in termini di tempo che di accuratezza, vengono riportati in questa tesi un insieme di risultati sperimentali ottenuti con un'applicazione multi-cluster.

ACKNOWLEDGEMENT

This master thesis has been carried out at the Department of Electronics and Information at Politecnico di Milano University. The work has been performed within System Architecture Engineering Group of professors Cristina Silvano and Vittorio Zaccaria who introduced the topic and provided limitless support during the course of the project.

The team work was truly fun and challenging at the same time. I learned a lot and met engineers who shared their knowledge and experience which I am very grateful and would like to thank.

First I would very much like to thank my supervisors at Politecnico di Milano, Vittorio Zaccaria and Cristina Silvano who always had the answer to all my questions and guided me to the right way. Their constant encouragement and support throughout the project made it possible for me to complete the work.

I would also like to thank Edoardo Paone, a co-worker in the System Architecture Engineering Group; it was a pleasure working with him and truly enjoyed his company.

I am not forgetting my best friends Mr. Kambiz Sethna, Miss Leila Nouri, and Mr. Ashkan Saidi Nejad who always been there.

Finally I would like to thank everyone who was involved in this project and helped me accomplish the goal of this thesis work.

Last but certainly not least I would like to thank my family for their unconditional love and support. Thank you for always being there for me; you are the true reason of my achievements in life.

Thank you all,

Nazanin Vahabi

TABLE OF CONTENT

CHAPTER 1

Introduction

Recently, multi-processor systems-on-chip (MPSoC) and chip multiprocessors (CMP) have become the fundamental computing frame of reference for application-specific processors. The current trend in System-on-Chip (SoC) design is to integrate a large number of processors, memories and hardware accelerators onto a single die, turning the concept of Multi Processor System-on-Chip into a reality.

MPSoCs are widely using platform-based design approach [1] which represents the best compromise in terms of hardware/software partitioning. This paradigm leads to risk reduction of missing the time-to-market deadline while ensuring greater efficiency by means of architecture customization and software compilation techniques.

It is often very difficult to find a single modeling approach or analysis tool which is capable of fulfilling all the challenges of multi-processor systems-on-chip design. Configurable simulation models are used to accurately tune the on-chip architectures and to fulfill the requirements of the target application in terms of performance, battery lifetime, and area.

The performance indicators (such as power consumption, delay, area, etc.) are impacted considerably by altering the parameters. The design space exploration (DSE) is an optimization phase which aims at tuning the configurable system parameters to find the best trade-off in terms of the selected figures of merit. The DSE generally consists of a multi objective optimization (MOO) problem consists of pruning a large design space of parameters at system and micro architectural levels.

The overall goal of the DSE phase is to find the optimal parameterized configurations of either architectures and/or applications in order to minimize the number of executing simulations during the exploration phase. So far, several heuristic techniques have been proposed to address this problem; however, they were not efficient enough for identifying the Pareto front of feasible solutions in a reasonable amount of time.

The methodology proposed by DSE technique to comply this goal is based on the design of experiments (DoE) and response surface modeling (RSM) techniques. An iterative process is repeated to derive the Pareto set. In each of the iterations, the DoE phase defines an initial plan of experiments to create a coarse grain view of the target design space. Continuously, a set of RSM techniques is used to refine the exploration and to identify the Pareto set consists of feasible configurations. Then, a technique is proposed to deal with the application-specific constraints expressed at the system level.

In most cases, the design space consists of huge number of configurations. In computer architecture, simulation represents the main tool to predict the performance of alternative architectural design points and is required to evaluate the configurations. In addition, evaluation of a single configuration almost always requires the use of simulators which are often time consuming. If we consider a cycle-accurate instruction set simulation, it needs a long time for simulation. Furthermore, the growth trend toward Chip Multi-Processor (CMP) architectures amplifies this problem because the simulation speed monotonically decreases by increasing the number of cores to be simulated. The DSE time increases linearly with the growth of the number of simulations to be done.

The main objective of the DSE strategy is to minimize the exploration time on one hand and to guarantee good-quality solutions on the other. The problem can be addressed in two complementary ways either by minimizing the number of configurations simulated, or by minimizing the time required to evaluate each configuration.

The work proposed in this thesis presents a tool for decreasing the time associated with simulations, or increasing the accuracy by using DoE and RSM techniques.

This document is organized as follows. Chapter 2 discusses the state of the art related to DSE. Then it continues describing in details the problem and the proposed solution. Chapter 3 introduces the Neural Network as a surrogate of simulations and I detail the analytical model used to find an optimal hierarchical topology. Chapter 4 validates the proposed methodology applied for the customization of a multi-cluster industrial architecture and investigates different training algorithms to find the best algorithm. Chapter 5 selects the best model for the neural network and chapter 6 shows the results of simulating with different simulators. Finally, Chapter 7 contains some concluding remarks.

CHAPTER 2

Background

In recent years, multi-processor systems-on-chip (MPSoC) have become more and more complex. One of the most important challenges of designing multi-processor systems-on-chip is the variety of design possibilities that need to be considered. The design space usually involves multiple metrics of interest (latency, resource usage, energy consumption, cost, etc.) and multiple design parameters (e.g. the number and type of processing cores, sizes and organization of memories, interconnections, scheduling and arbitration policies, etc.). It is often very difficult to find a single modeling approach or analysis tool which is capable of fulfilling all the challenges of multi-processor systems-on-chip design due to the relation between design choices on one hand and the metrics of interest on the other.

In the design line of System-on-Chip (SoC) platforms which are known as pre-designed parametric [1] architectural solutions, an architectural template is gradually refined step by step on the basis of functional specification and system requirements. With the term ‘‘platform’’ we mean a coordinated family of hardware/software architectures developed to highly reuse hardware and software components in design of application-oriented derivative products which have to be rapid and at the same time low risk. In this context, platform-based design approach [1] represents the best compromise in terms of hardware/software partitioning. This property has led to a reduction in the time-to-market while ensuring greater efficiency by means of architecture customization and software compilation techniques.

2.1 State of the Art

Design Space Exploration (DSE) strategy aims at minimizing the exploration time on one hand and guaranteeing good-quality solutions on the other. The problem can be addressed in two complementary ways either by minimizing the number of configurations simulated, or by minimizing the time required to evaluate each configuration. Givargis et al in [2] focus on the concept of parameters dependency. Their approach is based on clustering dependent parameters and then carrying out an exhaustive exploration

within these clusters. If the size of these clusters increases too much due to great dependency between the parameters, the approach becomes a thorough search, with a consequent loss of efficiency. Their approach guarantees the quality but does not reduce the exploration space. Another category of approaches in the literature aims at reducing the number of configurations however they are not the optimal. Fornaciari et al. in [3] use sensitivity analysis to reduce the exploration space from the product of the cardinalities of the sets of variation of the parameters to their sum. This approach can be seen as a simplified version of [2], as all the parameters are considered to be independent. In [4-7] other DSE approaches are proposed which perform the pruning of the design space. Most of the approaches belonging to the second category are of limited applicability and not general (or scalable) since they are often tailored for a specific system architecture. Ghosh et al. use an analytical model to speed up evaluation of a system configuration in [8]. Statistical simulation is used in [9] to speed up the evaluation of configurations by means of a multi- objective genetic algorithm.

Among other heuristics to reduce the DSE complexity, in [10] the authors compare Pareto simulated annealing and random search exploration to find a good approximation of the Pareto-optimal configurations representing the best energy-delay trade-offs by varying the architectural parameters of the target super scalar architecture executing a set of multimedia kernels. In [1], Ascia et al. propose the use of Multi-objective Evolutionary Algorithms as optimization technique and Fuzzy Systems for the estimation of the performance indexes to be optimized. The technique is applied to a highly parameterized SoC platform based on a very long instruction word processor in order to optimize both power dissipation and execution time. The technique is based on a strength Pareto evolutionary algorithm coupled with fuzzy system rules in order to speed up the exploration phase and improve the accuracy.

Further recent system performance optimization is presented in [11-14]. All of the proposed methods use a combination of RSM and DoEs methodologies.

In [11], Joseph et al. propose an approach in which an initial training sample set of the whole design space is selected in order to obtain a good estimation accuracy. The performance of a superscalar architecture is estimated by a radial basis function (RBF) and the optimal initial set is derived from the Latin hypercube method.

In [12] and [13], linear regression has been used for the performance prediction and assessment. In [14], McKee et al. try to have a better performance prediction by using an Artificial Neural Network (ANN) paradigm to estimate the system performance of a Chip Multi Processor (CMP).

The methodology proposed in [15] is the extended version of the [16] and [17]. All these papers represent an efficient DSE methodology with leveraging traditional Design of Experiments (DoE) and Response Surface Modeling (RSM) techniques. [15] in addition proposes an overall generalization and formalization of the proposed exploration strategy based on several RSMs and a refined technique to deal with the application-specific constraints.

This thesis benefits from the methodologies above, and tries to use Neural Network to model the configurations simulation of a customizable OPENCL Stereo-Matching application for a multi-cluster, industrial architecture.

2.2 Introduction to Design Space Exploration

As mentioned above, the performance indicators (such as power consumption, area, etc.) are impacted considerably by altering the parameters. Defining strategies to tune parameters in order to obtain the optimal system configuration represents a challenge known as Design Space Exploration (DSE). In other words, DSE consists of solving a multi-objective optimization (MOO) problem by pruning a large design space of parameters. The overall goal of the DSE phase is to find the optimal parameterized configurations of either architectures and/or applications to acquire the optimal system-level requirements depending on the given application. So far several heuristic techniques have been proposed to address this problem; however, in some cases they are not efficient enough for identifying the Pareto front of feasible solutions in a reasonable amount of time. Among those heuristics, evolutionary or sensitivity-based algorithms represent a remarkable role in state-of-the-art techniques.

Any DSE technique can be schematically represented as in Figure 2.1 [1]. The exploration flow starts with a base configuration and iteratively improves the starting configuration. The process includes two main stages: evaluation and tuning of the parameters of the configurations. The evaluation phase often encapsulates system-level simulation which forms an obstacle in the exploration process. The tuning phase uses the results of the evaluation phase to modify the system configuration parameters in order to optimize determined performance indicators. The loop ends when a system configuration that meets the design constraints has been obtained, or, more frequently, when a set of Pareto-optimal configurations for the indexes to be optimized have been accumulated.

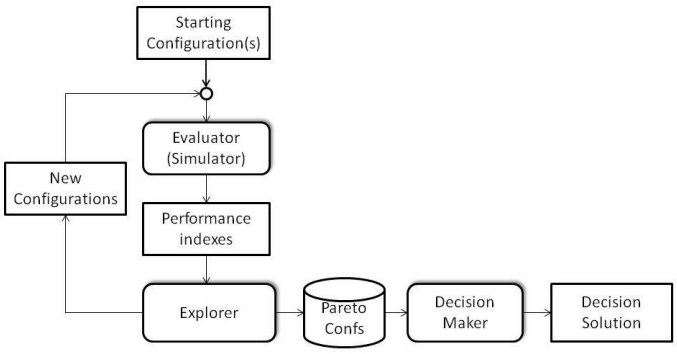

Figure 2.1

Design Space Exploration Flow

In most cases, the design space consists of huge number of configurations. In addition, evaluation of a single configuration almost always requires the use of simulators which are often time consuming. Another problem is that the optimized objectives are often conflicting. The result of the exploration will therefore not be a single solution but a set of trade-offs which make up the Pareto set.

As explained before, in computer architecture, simulation represents the main tool to predict the performance of alternative architectural design points and is required to evaluate the configurations. If we consider a cycle-accurate instruction set simulation, it needs a long time for simulation. Furthermore, the growth trend toward Chip Multi-Processor (CMP) architectures amplifies this problem because the simulation speed monotonically decreases by increasing the number of cores to be simulated [15].

2.3 Problem

In this thesis we address the problem of the time consumption for simulation of an application targeted to P2012. Platform 2012 consists of a many-core computing fabric with a noticeable area/power efficiency which makes the integration of hardwired accelerators easier by means of an architectural harness [18]. P2012 is designed and produced by STMicroelectronics. The P2012 project is based on multiple processor clusters and thus, it is highly modular. The P2012 project aims at moving a significant step forward in programmable accelerator architectures for next-generation data-intensive embedded applications such as multi-modal sensor fusion, image understanding and mobile augmented reality.

Achieving performance and platform portability, P2012 supports different programming models. Keep following the software point of view, Standards-based Programming Models which are based on industrial standards can be implemented effectively on the P2012 platform. The programming language OpenCL is supported since 2011, and OpenMP support is planned for 2012. The Native Programming Model (NPM) is closely coupled to the platform and provides the highest level of control on application-to-resource mapping, at the expense of abstraction. Advanced Programming Models (typically data flow variations) are tuned to exploit combinations of HW and SW processing elements within a cluster. P2012 programming tools assist the developer from high-level application capture and simulation, to analysis, debugging and visualization of the performance- and power-optimized version of the application mapped onto the fabric, including its interaction with a host "full stop" processor Programming-model awareness is maintained at all abstraction levels and for all the different tools provided[18].

Currently P2012 is still in prototyping; therefore we are forced to work with a simulator. However, STMicroelectronics provides a simulator (Gepop) that allows for simulating P2012 at different abstraction layers. One of the simulation models provided by Gepop is called "posix-xp70". In this model the code executed on the P2012 cluster is simulated by means of an Instruction Set Simulator (ISS), which gives cycle-accurate information for the execution time. Instruction set simulator (ISS) is a simulation model, usually coded in a high-level programming language, which mimics the behavior of a mainframe or microprocessor by "reading" instructions and maintaining internal variables which represent the processor's registers. The experimental data set is derived from the simulation on the "STM Gepop posix-xp70 model" which will be referred to as "posix-xp70" in the rest of this thesis.

Instruction simulation is a methodology employed for several reasons. One of these possible reasons is to improve the speed performance - compared to a slower cycle-accurate simulator - of simulations involving a processor core where the processor itself is not one of the elements being verified.

Our target application implements the Stereo Matching algorithm for image processing proposed in [19]. Stereo Matching is one of the important vision problems which estimates disparities from a given stereo image pair. To reduce the image ambiguity, local stereo matching methods are commonly used. We use an area-based local stereo matching application for accurate disparity estimation across all image regions.

In this thesis, the Stereo Matching application has a set of input parameters. By changing the values of parameters, some application metrics are affected, such as the delay (execution time) and the accuracy of result (average error per pixel) in our case. In fact, we are interested in those configurations that represent optimal trade-offs between application metrics. A method for solving this Multi-Objective Optimization (MOO) problem is to consider only the configurations on the Pareto frontier. In general, Design Space

Exploration considers a set of possible configurations of parameters that executes for application profiling and finally collects the results.

2.4 Solution

The ISS takes a long time to simulate the Stereo Matching algorithm on one configuration. The simulation process on one image takes 30 minutes. In one configuration we process 7 different images to compute the average metric values over a variable set, thus simulation process of one configuration takes four hours. Therefore collecting the data would be a time consuming process which takes weeks or months (depending on the machine) of simulation. To overcome the long simulation time, we propose a methodology which is based on minimizing the number of executing simulations during the design space exploration phase. The methodology is based on the Design of Experiments (DoE) and Response Surface Modeling (RSM) techniques. In DoE phase we create a coarse grain view of the target design space by defining an initial plan of experiments. Each DoE plan differs in terms of the layout of the selected design points in the design space. Moreover, a set of RSM techniques is used to identify a set of feasible configurations and improve the exploration. This process is iteratively repeated to derive a set of Pareto points [15].

The term DoE [20] is used when the effects of tuning a set of variable parameters on information-gathering experiments are examined.

DoE focuses on the effects of some parameter's tuning on the system response. By changing the sets of parameters, some application metrics are affected, such as the delay (execution time) and the accuracy of result (average error per pixel). DoE is a set of techniques whose main goal is the screening and analysis of the system behavior with a small number of simulations.

To reduce the number of simulations it is possible to use RSMs that allow for prediction of the values of application metrics without running the application (of course with some error). RSM techniques are typically introduced to decrease the time due to the evaluation of the system-level objective function f(x) for each architecture x [15].In fact, f(x) evaluation includes one or more simulations which can take several hours. However, it depends on the system resources of the simulation and complexity of the platform.

A response surface model for the function f(x) is an analytical function r(x) such that:

$$f(x)= r(x)+ \varepsilon \quad [8] \qquad 2.1$$

where ε is the estimation error. Typically, a suitable RSM for f(x) has to be chosen in a way that ε has some desired statistical properties such as a mean of zero and small variance. The principle of RSM is using a set of simulations generated by DoE in order to build our response model of the system. A typical RSM-based flow involves a training phase, in which known data (or training sets) are used to learn RSM configuration, and a prediction phase, in which the RSM is used to forecast the unknown system response. RSMs are an effective technique for analytically predicting the behavior of the system platform without resorting to a system simulation.

2.5 Target Exploration Strategy

Palermo et al. proposed an exploration strategy which is called response surface-based Pareto iterative refinement (ReSPIR) [15]. A key concept of the proposed methodology is how the Pareto sets are built. The approach concentrates on the concept of iterative simulation-based refinements of the approximate Pareto set which is obtained from the RSM model predictions, starting from an initial DoE (Figure 2.2).

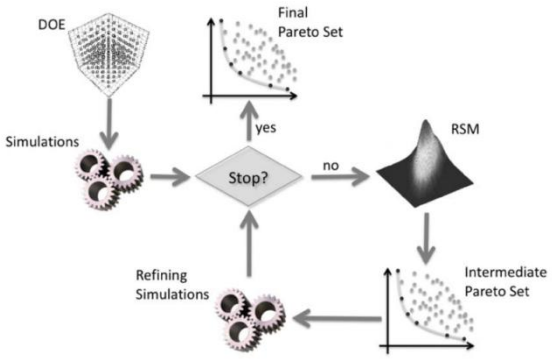

Figure 2.2 ReSPIR DSE flow[15]

The methodology is parametric in terms of DoE and RSM techniques to be used, as well as in terms of the maximum number of simulations to be run. Therefore, the methodology which is proposed in this thesis can be used as the RSM technique in ReSPIR methodology [15].

2.6 Response Surface Model Selection

As explained in Section 2.3, we introduce RSM to provide an analytical representation r(x) of the given vector objective function f(x). In this thesis, we propose an approach which tackles the design space exploration (DSE) problem in both fronts of the reduction in number of system configurations to be simulated and the reduction of the time required to evaluate (i.e., simulate) a system configuration. RSM-based techniques represent the kernel of the proposed methodology. Following Palermo et al. in [15], in this thesis, Artificial Neural Network as one of the RSM techniques is proposed to aim the problem and tuned to build the response surface models, thus, speed up the overall exploration phase.

An ANN, usually called Neural Network (NN), is a computational model that is inspired by the structure and functional aspects of biological neural networks. As it is shown in figure 2.1, a neural network consists of an interconnected group of artificial neurons. In most cases an ANN is an adaptive system that changes its structure based on external or internal information which flows through the network during the learning phase. Modern neural networks are commonly used to model complex relationships between inputs and outputs or to find patterns in data.

Perhaps the greatest advantage of ANNs is their ability to be used as an arbitrary function approximation mechanism that 'learns' from observed data.

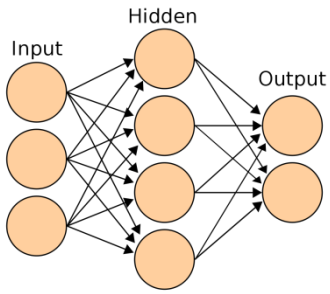

Figure 2.3 An artificial neural network is an interconnected group of nodes, related to the vast network of neurons in the human brain.

ANNs are machine learning models that automatically learn to predict targets (in our case, simulation results) from a set of inputs. ANNs constitute a powerful, flexible method for generalized nonlinear regression, and deliver accurate results in the presence of noisy input data [14].

The Artificial Neural Network with its advantages will be considered and utilized on the design space configurations. The idea of this type of modeling is to find a pattern in the training data that can be adopted when required to predict the behavior of unknown configurations. In fact, we propose a design space exploration methodology minding the statistical behavior of known configurations to predict the unknown configurations properties, by means of analyzing via simulations. By implementing this method, we guarantee that the model can be modified easily when any change is applied in the future for the system.

Another goal of the proposed methodology is to address the training data set in terms of size. On one hand, the big number of training data leads to learn better the system statistical parameters and, therefore, more precised predictions. On the other hand, each configuration will take a long time to simulate. The synergy between custom design space size and the configuration simulation time is supposed to play a significant role in improving the target figures of merit. In fact, the knowledge of few design points is used to predict the expected output of unknown configurations. We propose that the correlation of the configurations within the multi-processor design space can be modeled successfully with neural network training functions to accelerate the exploration phase.

In the heuristics, NNs have been used in research and commercially to guide autonomous vehicles [21], to play backgammon [22] and to predict weather [23], stock prices, medical outcomes, and horse races. The representational power of ANNs is rich enough to express complex interactions among variables. Any function can be approximated to arbitrary precision by a three-layer ANN [24]. We selected ANNs over other predictive models such as linear or polynomial regression and Support Vector Machines (SVMs) for modeling parameter spaces in computer architecture due to [14]:

1. They operate with Real, Discrete, Cardinal, and Boolean valued inputs and outputs, and thus can represent parameters of interest to an architect.

2. They work well with noisy data, and thus can successfully be combined with existing mechanisms that reduce the time that simulation experiments take at the expense of introducing noise.

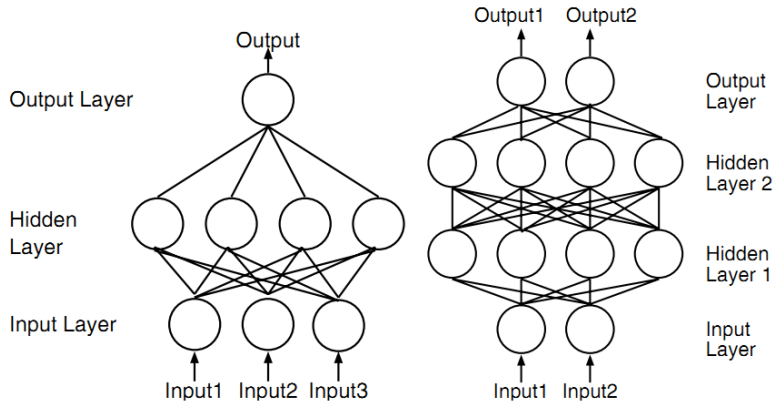

Figure 2.4 Simplified Diagram of fully connected, feed-forward ANN [10]

Figure 2.4 shows the basic organization of simple fully connected, feed-forward ANNs. It depicts two networks, one of them consists of an input layer, output layer, and one hidden layer; and the other one has more hidden layers. Input values are presented at the input layer; predictions are obtained from the output layer. Each unit operates on its inputs to produce an output that passes to the next layer. In fully connected feed-forward ANNs, the units of each layer are connected to all the units of the next layer by weighted edges, communicating outputs to other units downstream. A unit applies its activation function to the weighted sum (based on edge weights) of all inputs.

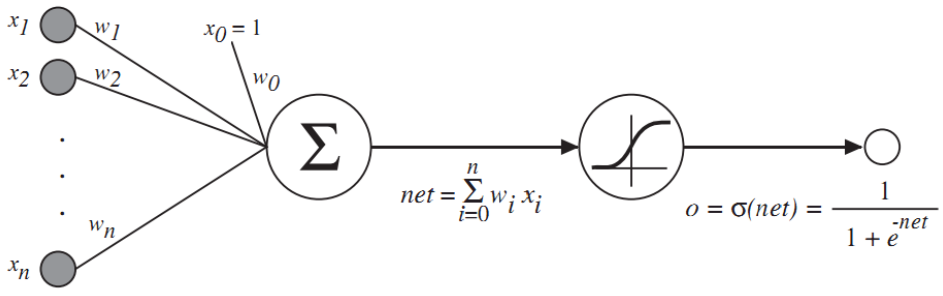

Figure 2.5 Example of a hidden unit with a sigmoid activation function [22]

Figure 2.5 reveals a hidden unit using a sigmoid activation function. In general, activation functions must be nonlinear, monotonic, and differentiable but being a sigmoid function is not necessary. Our models use sigmoid activation functions.

Moving towards many-core architectures opens up many opportunities for developing new optimization phases. Design Space Exploration can help accurately choosing the most promising configurations to be analyzed with a neural network simulation model.

CHAPTER 3

Neural Network Modeling of an Application Running on STMicroelectronics P2012 Architecture

3.1 Main Objectives

Generally speaking, in computer architecture, simulation represents the main tool to evaluate the configurations. As it is explained above, a cycle-accurate instruction set simulator requires a long time simulation. The RSM-based techniques provide an analytical representation of the configurations. As mentioned before, ANNs tackle the problem of design space exploration (DSE) by reducing the time required to evaluate a system configuration. ANN aims at building the response surface models, thus, speeds up the overall exploration phase. ANN represents a powerful and flexible method for generalized nonlinear regression.

In the next chapters we try to validate and support statistically the use of Neural Network as a surrogate of simulations. To validate the proposed ANN response surface modeling methodology, we applied it to the customization of the OPENCL Stereo-Matching Application for a multi-cluster industrial architecture.

There is a set of input parameters which affects some metrics of the Stereo-Matching application, such as the accuracy of result (disparity error). Using ANN to model the simulations allows for exploration of all possible configurations of input parameters and analysis of the metrics measured for each configuration. We are actually interested only in those configurations that represent optimal trade-offs between application metrics. A method for solving this multi-objective optimization problem is to consider only the configurations on the Pareto frontier.

In the Stereo-Matching implementation, there are some parameters (nb_wta_workgroups, wg_col_pixel_width, nb_wi_per_wg and nb_hypo_per_wg) that are called resource parameters. By changing these parameters, the application-specific metrics (such as the accuracy of result) does not change; only the execution time is affected by changing resource parameters. These parameters should be configured according to the specific target architecture, keeping into account some platform

characteristics such as cache size, memory structure, number of processing elements, etc. However, OpenCL is a cross-platform standard, so the same code can be executed on different target platforms.

This makes an important goal which is optimizing the design for a specific platform by means of application customization. This means that the same application can be ported to different target architectures (e.g. x86 multi-core processor or P2012 many-core computing fabric) by selecting the configurations of application parameters which work well on the specific architecture.

In general, Design Space Exploration considers a set of possible configurations of parameters, executes the application for each of them and finally collects the results. The target architecture is the one already presented in Section 2.3. In this simulation model the code executed on the P2012 cluster and configurations are simulated by means of the Instruction Set Simulator on the posix-xp70 simulation model.

Therefore, our multi objective problem presents the process of finding a system configuration which minimizes the average value of the system response in each single application scenario. In particular, we formalize our multi objective problem as a minimization of the average execution time and disparity error system response. We train the NN with a set of configurations and eventually study the system response in different scenarios to find the most fitted architecture parameters to achieve an acceptable performance.

Neural network is affected by its parameters such as the initial data set, the number of iteration steps, the set of training functions, the number of hidden layers/ neurons in each layer, and other specific parameters.

3.2 Initial data set, gathered from simulations

To reach the goal, we require samples from design space as the input for the neural network. We collected the input data by simulating the configurations on Gepop posix-xp70.

To provide a comprehensive validation, the analysis focuses on the architectural parameters listed in Table 3.1, representing a design space composed of $|X| = 3 * 18 * 3 * 4 = 648$ configurations to be simulated by posix-xp70. From these 648 configurations, the database contains only 605 valid configurations. The remaining 43 configurations are not feasible on the target device. The reason is that the memory available on the device is not enough to allocate the buffers required by these configurations.

Parameter	Possible Values
hypo_step	1-2-3
confidence	44
max_arm_length	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18
match_limit	60
max_hypo_value	200
nb_wi_per_wg	16
nb_hypo_per_wg	1-2-4
nb_wta_workgroups	1
wg_col_pixels_width	48-64-80-96

Table 3.1 The Input Parameters

The possible values for each parameter have been set as shown in Table 3.1. It should be noted that some of the parameters are constant and are not considered as input data.

The actual system response (represented in Table 3.2) that consists of the average execution time (wall-time) is measured by means of the posix-xp70 profiler (based on ISS), while the disparity error (disper) is computed from the application. We take into account only execution time and disparity error because the current version of posix-xp70 does not profile energy consumption.

Wall_time is the time required to execute the OpenCL kernels on the P2012 device (simulated by an ISS). Local_mem_usage is the amount of L2 memory (memory on the P2012 cluster) allocated for Stereo Matching kernels. Disper is the average error per pixel. The result of the application is a disparity map, which is compared to a reference map (what we call the truth map) to calculate this error. It has to be considered that the total memory available on cluster is 256KB= (256*1024) Bytes).

Parameter	Description
Wall_Time	Kernel Execution Time
Disper	Result error / The opposite of accuracy
local_mem_usage	The memory used by Multi View in Bytes

Table 3.2 The Output Parameters

3.3 The Neural Network software

The Neural Network Toolbox integrated in MATLAB® provides tools for design, implementation, visualization, and simulation of neural networks. The model of the design space configuration was built by using Matlab ®Neural Network Toolbox R2007b. Matlab is a powerful software that allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran. An additional package, Simulink, adds graphical multi-domain simulation and Model-Based Design for dynamic and embedded systems.

Neural networks are used for applications where formal analysis such as pattern recognition and nonlinear system identification and control would be difficult or impossible to perform. Neural Network Toolbox supports feed-forward networks, radial basis networks, dynamic networks, self-organizing maps, and other proven network paradigms.

3.4 Training Algorithm

Machine learning models require some type of training experience from which to learn. Here, training examples of the design space simulation results are used. Training an ANN involves learning edge weights from these examples. The edge weights of an ANN define the functional relationship between input and output values. In order to predict the execution time and disparity error, the architect runs a number of cycle-by-cycle simulations for combinations of parameters, collecting the parameters and resulting execution times and accuracy errors into a training dataset. The weights are adjusted based on these data until the ANN accurately predicts the outputs from the input parameters. Obviously, a good model must make accurate predictions for parameter combinations on which it was not trained.

There are different training algorithms which can be utilized during the training phase. Selecting different algorithms affects the metrics prediction. The difference between the training functions is that they update weight and bias values according to different algorithms. In our case, the three most fitted training functions were applied on the neural network and the performance of the NN was evaluated in terms of Mean Squared Error and Root Mean Squared Error. The goal of this part is to select the best training function which leads to the minimum training error between the predicted configurations and simulated configurations.

As we use a random DoE and a neural network (e.g., the learning set is chosen randomly from the training set), each combination of heuristic parameters should be evaluated more than once to infer a more general trend. The number of runs for each algorithm is set such that the actual performance of the algorithm (in terms of Root Mean Squared Error) reaches an average asymptotic value. This generated more than 100 evaluations for each heuristic.

Each time the neural network was trained with another training function by considering an upper bound on the number of training data which is 90% of the complete design space starting from 50%, since we are focused on obtaining an approximate algorithm.

3.5 Training Set Size

Similar to other regression methods, ANNs learn less accurate models from reduced training samples. On the contrary, the big number of training samples means more configurations to be simulated. However, data collection in architecture design space exploration is expensive, and a trade-off exists between number of simulations and model accuracy. The smaller training data set with an acceptable NN performance allows us to estimate model accuracy and execution time. The overall goal is to minimize the number of simulations to be executed during the exploration phase. One way to reach this goal is to minimize the number of training samples which have to be simulated. The number of training samples is one of the complexity parameters of the DSE phase. During this phase, we also take into account that one of the ANN parameters that most impact learning is the number of hidden layers and number of hidden neurons per layer. Finding the optimal settings that perform well is typically straight forward. We evaluated the neural networks performance by applying 1 to 5 hidden layers, and 1 to 5 hidden neurons.

In this phase, the training function which was selected in the previous phase has been set. This time the neural network was trained with different number of layers and neurons. Each strategy has been run by considering an upper bound on the number of training data which is 50% of the whole design space starting from 1%, since we concentrate on obtaining an approximate training data set size (RMSE≤20%) by executing less than one fifth times of the total combinations of layers and neurons.

The number of runs for each algorithm is set to 10 such that the actual performance of the algorithm (in terms of Root Mean Squared Error) reaches an average asymptotic value.

Thus, we pruned accordingly the parameter space of each heuristic to obtain those configurations that leads to a suitable number of simulations. The resulting Pareto front has been validated against the reference exact Pareto front of the target architecture.

The neural network is used to model the platform 2012 architecture. In this simulation model the configurations are simulated by means of the instruction set simulator on the posix-xp70 simulation model. To initialize the neural network we do need to find the best fitting for its parameters such as training algorithm, training set size and the number of layers and neurons which affect the neural network behavior and prediction.

CHAPTER 4

Selection of the Training Algorithm for the Neural Network

The methodology we have proposed, aims to accelerate the process of simulating the numerous design space configurations employing multi-processor design space within Artificial Neural Network functions.

In this chapter, we show the experimental results obtained by applying this methodology to the customization of the OPENCL Stereo-Matching for a multi-cluster industrial architecture case study.

4.1 System Modeling and Simulations

This section explains how the design space configuration model was built using Matlab ®Neural Network Toolbox.

The intention of this modeling system is find and adopt the specific pattern in the training data when it is required to predict the behavior of unknown data. Therefore implementation of this model would guarantee that any possible further changes in the system will be easily modified.

Regarding this issue, we need to create a new feed-forward back propagation neural network by means of newff syntax.

The parameters involve in newff are:

- Input and output data set which are imported to the neural network.
- Numbers of layers and neurons
- Training function

As we mentioned in the previous sections, a neural network is an interconnected group of neurons, akin to the vast network layers. The simplest existing network is single-layer neural network with one neuron. The networks, we examined in this dissertation are single to multi-layers (1 to 5) with 1 to 5 neurons.

To train each neural network, it is required to set the proper training functions. Even though the default training function of Matlab – Trainlm - was fast enough for training process but we did examine all the neural network training functions in order to find the most suitable ones. Afterwards Trainlm, Trainrp, and Trainbfg were selected as the best solutions for these neural networks. These training functions will be explained later on in more details.

Generally to execute a neural network, each network's weights and biases should be configured and initialized. Once these procedures are performed the network is ready for training.

In the training process, network learns from the training data set which consists of simulated configuration samples. Primarily, in this process neural network splits the data set in to training, validation and test data.

One of the selecting methods is randomly picking the training data. In this section, we gave different range of data set (simulated configuration samples) from 50% to 90% to training data in order to discover the best solution with the minimum error and maximum accuracy in predicting the behavior of neural network.

The state-of-the-art techniques incur in long simulation time to evaluate a comprehensive subset of the design space. In fact, simulation can take several days, depending on the application complexity. Here, we study the prediction error with applying different number of training data. The goal is to optimize the prediction in terms of delay and accuracy. In fact, the knowledge of few design points is used to predict the expected improvement of unknown configurations.

4.2 Different Training Functions

Trainlm is a network training function which updates weight and bias values according to Levenberg-Marquardt optimization algorithm [25]. Although, the Levenberg-Marquardt (LM) algorithm is the most widely used optimization algorithm, this method is not efficient enough in terms of memory and time especially when large number of training patterns need to be considered. Accordingly in order to address these problems recent methods have been presented for saving significant amount of computation memory as well as increasing the overall performance of the LM method.

The LM algorithm is the first shown to be a blend of vanilla gradient descent and Gauss-Newton iteration.

Trainlm (net,Pd,Tl,Ai,Q,TS,VV,TV) takes these inputs:

Net	Neural Network
Pd	Delayed input vectors
Tl	Layer target vectors
Ai	Initial input delay conditions
Q	Batch size
TS	Time steps
VV	Either an empty matrix [] or a structure of validation vectors
TV	Either an empty matrix [] or a structure of test vectors

And it returns the following outputs:

Net	Trained network
TR	Training record of various values over each epoch:
TR.epoch	Epoch number
TR.perf	Training performance
TR.vperf	Validation performance
TR.tperf	Test performance
TR.mu	Adaptive mu value

Unlike other training functions, trainlm assumes that the network has the Mean Squared Error (MSE) performance function. This is a basic assumption of the Levenberg-Marquardt algorithm. The MSE is explained in the further paragraphs.

Trainrp is a network training function that updates weight and bias values according to the resilient back propagation algorithm (Rprop). The Rprop algorithm proposed by Riedmiller and Braun is one of the best performing first-order learning methods for neural networks. A common and quite general method for improving network training is weight- back tracking. Weight-back tracking means retracting a previous weight update for some or all weights. Rprop is a learning algorithm for multilayer feed-forward networks. To overcome the inherent disadvantages of pure gradient-descent, Rprop performs a local adaptation of the weight-updates according to the behavior of the error function. Contrary to other adaptive techniques, the effect of the Rprop adaptation process is not blurred by the unforeseeable

influence of the size of the derivative, but only dependent on the temporal behavior of its sign. This leads to an efficient and transparent adaptation process [26].

Trainrp (net,Pd,Tl,Ai,Q,TS,VV,TV) takes these inputs:

Net	Neural Network
Pd	Delayed input vectors
Tl	Layer target vectors
Ai	Initial input delay conditions
Q	Batch size
TS	Time steps
VV	Either an empty matrix [] or a structure of validation vectors
TV	Either an empty matrix [] or a structure of test vectors

And it returns these outputs:

Net	Trained network
Ac	Collective layer outputs for last epoch
El	Layer errors for last epoch
TR	Training record of various values over each epoch:
TR.epoch	Epoch number
TR.perf	Training performance
TR.vperf	Validation performance
TR.tperf	Test performance

Trainbfg is a network training function that updates weight and bias values according to the BFGS quasi-Newton method. Newton's method is an alternative to the conjugate gradient methods for fast and large scale optimization. BFGS quasi-Newton algorithm is fitted for constructing a single hidden layer feed-forward neural network. Usage of quasi-Newton as a method to minimize the sequence of error functions of growing network is one of the most significant feature of this algorithm. Basically, based on experimental results this algorithm is highly robust and efficient [27].

The trainbfg (net,Pd,Tl,Ai,Q,TS,VV,TV) takes these inputs:

Net	Neural Network
Pd	Delayed input vectors
Tl	Layer target vectors
Ai	Initial input delay conditions
Q	Batch size
TS	Time steps
VV	Either an empty matrix [] or a structure of validation vectors
TV	Either an empty matrix [] or a structure of test vectors

And it calculates the following outputs:

Net	Trained network
Ac	Collective layer outputs for last epoch
El	Layer errors for last epoch
TR	Training record of various values over each epoch:
TR.epoch	Epoch number
TR.perf	Training performance
TR.vperf	Validation performance
TR.tperf	Test performance

4.3 Preliminary Definitions

The proposed methodology uses both Mean (MSE) and Root Mean Squared Error (RMSE) to evaluate the prediction performance. The MSE of a simulator is used, in order to quantify the difference between predicted values implied by a simulator and the true values of the quantity. Moreover, MSE measures the average of the squares of the "errors." The error is the amount by which the value predicted by the neural network differs from the quantity which is implied by the configuration itself. The MSE of an estimator with respect to the estimated parameter is defined as

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{E}\left[(\hat{\theta} - \theta)^2\right]. \qquad 4.1$$

RMSE is a frequently used in order to measure the differences between values predicted by a model or by an estimator and the actual observed value. RMSE (sometimes is called Root Mean Square Deviation (RMSD)) is an efficient measurement tool with high accuracy. Regarding the estimated parameter the RMSD is defined as the square root of mean square error

$$\mathrm{RMSD}(\hat{\theta}) = \sqrt{\mathrm{MSE}(\hat{\theta})} = \sqrt{\mathrm{E}((\hat{\theta} - \theta)^2)}. \qquad 4.2$$

4.4 The Best Training Function

In this study we focus on the iterative modeling to compare the predicted output and simulated output. This method calculates the prediction error for each training function and chooses the one with the least prediction error.

To initialize this algorithm, it is essential to set a new neural network considering the following parameters: number of layers and neurons, training function and the percentage of training data.

Primarily, the neural network is weighted with one neuron and one layer and 50% is set as the training data. In each of the iterations, the algorithm identifies a set of configurations for training and for predicting its output. Furthermore, this algorithm compares the predicted output with the expected target output to calculate the MSE and RMSE.

Number of neurons and layers is increased one by one in the further steps to cover 25 possible combinations. In other hand, training data percentage is also increased 10% by 10% until it covers 90% of training data percentage.

The error calculation is used to find validation and test output as well. But, in this section we just focus on training error estimation as a method to select the best training function.

4.4.1 Design of Experiments

Taking into account the impact of parameter interaction on the system-level metrics, an extension of the Box-Behnken DoE is selected among the others [28]. The Box–Behnken DoE allows generating experiments to capture those interactions. Thus, we selected this DoE to build our efficient DSE methodology. The Box–Behnken design is suitable for quadratic models where parameter combinations are at the center of the edges of the process space in addition to a design with all the parameters at the center. This leads to avoidance of taking extreme values for all the parameter combinations at the same time and it counts as a remarkable advantage. Because this avoids singular points in the generation of the response surface which pulls it down.

Considering all the possible combinations of number of layers and neurons, a Box–Behnken design is composed of the following design vectors (<number of layers, number of neurons>):

{<1,3>, <2,3>,<3,3>,<4,3>,<5,3>,<3,1>,<3,2>,<3,4>,<3,5>}

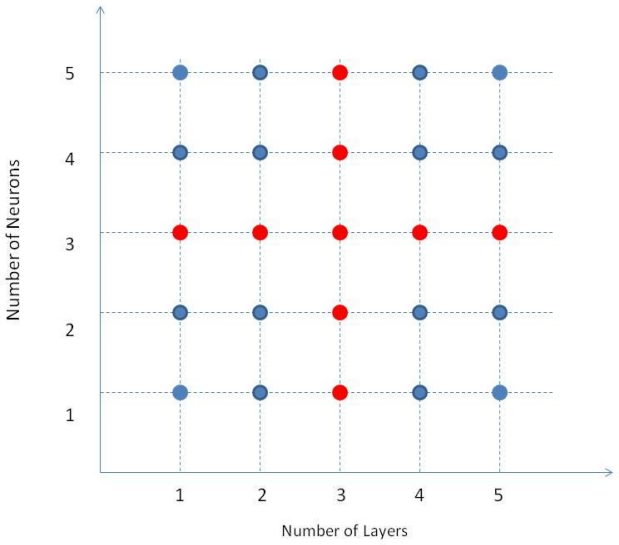

Figure 4.1 The red points show the 9 samples which have been selected out of the complete set of 25 samples by an extension of Box-Behnken Design statistical sampling technique.

4.4.2 Experimental Results of the Execution Time Network

The first network model intends to predict the execution time (wall-time). In this section we focus on the algorithm which is executed by MATLAB neural network toolbox with 45 iterations. The RMSEs which are calculated in iterations are the root mean square errors between the predicted execution time values during the training and the true execution time values.

The first function set as training function is trainlm and the training data set is selected randomly through the entire design space.

	Training data set Percentage				
(layers,neurons)	0.5	0.6	0.7	0.8	0.9
(3,1)	0.08	0.68	0.16	0.04	0.19
(3,2)	0.14	0.17	0.17	0.03	0.09
(3,3)	0.04	0.17	0.17	0.17	0.17
(3,4)	0.06	0.16	0.06	0.04	0.04
(3,5)	0.19	0.06	0.08	0.17	0.17
(1,3)	0.18	0.28	0.17	0.28	0.3
(2,3)	0.17	0.17	0.07	0.07	0.17
(3,3)	0.04	0.17	0.17	0.17	0.17
(4,3)	0.04	0.14	0.04	0.19	0.04
(5,3)	0.03	0.08	0.03	0.1	0.03

Table 4.1 The RMSEs between the predicted execution time values and the observed values during the simulation calculated by the network which is trained by trainlm function.

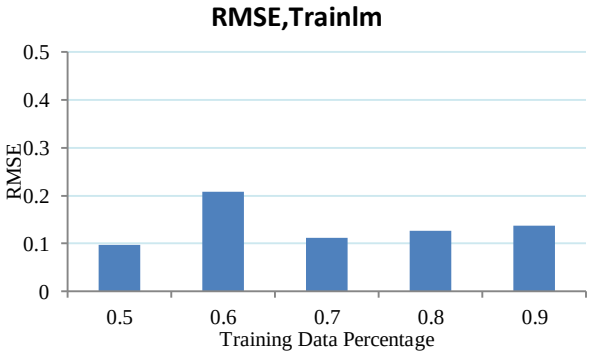

Figure 4.2 The RMSE values over the training data percentage during the simulation calculated by the network which is trained by trainlm function.

Table4.1 illustrates the RMSE values, which has been trained by trainlm and intends to predict the execution time (wall_time). The presented RMSEs have been calculated based on the selected layers and neurons in previous chapters.

In figure 4.2, the bar chart presents the same information about the RMSE values calculated by trainlm transfer function of different training data set from (50% to 90%).

Next, the trainrp is set as the training function and all the levels are repeated to calculate the RMSEs between the predicted execution time and observed execution time of the first network when it trains by trainrp function.

	Training data set Percentage				
(layers,neurons)	0.5	0.6	0.7	0.8	0.9
(3,1)	0.19	0.07	0.17	0.18	0.18
(3,2)	0.1	0.18	0.2	0.1	0.18
(3,3)	0.17	0.08	0.12	0.2	0.11
(3,4)	0.14	0.18	0.17	0.18	0.1
(3,5)	0.08	0.08	0.17	0.17	0.18
(1,3)	0.19	0.2	0.18	0.2	0.18
(2,3)	0.1	0.18	0.18	0.2	0.18
(3,3)	0.17	0.08	0.12	0.2	0.11
(4,3)	0.05	0.08	0.08	0.2	0.17
(5,3)	0.06	0.11	0.07	0.07	0.06

Table 4.2 The RMSEs between the predicted execution time values and the observed values during the simulation calculated by the network which is trained by trainrp function.

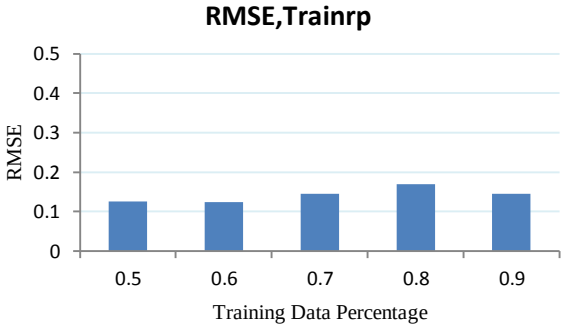

Figure 4.3 The RMSE values over the training data percentage during the simulation calculated by the network which is trained by trainrp function.

Table 4.2 and 4.3, illustrate the RMSE values which are calculated by the neural networks that predicts execution time, these values have been trained by trainrp and trainbfg functions respectively. In figures 4.3 and 4.4, the bar charts present the same information about the RMSE values calculated by trainrp and trainbfg transfer functions of different training data percentage from (50% to 90%).

	Training data set Percentage				
(layers,neurons)	0.5	0.6	0.7	0.8	0.9
(3,1)	0.07	0.18	0.18	0.23	0.20
(3,2)	0.17	0.17	0.05	0.18	0.19
(3,3)	0.04	0.17	0.05	0.07	0.19
(3,4)	0.09	0.19	0.07	0.06	0.18
(3,5)	0.07	0.16	0.16	0.14	0.17
(1,3)	0.18	0.17	0.18	0.19	0.19
(2,3)	0.07	0.17	0.08	0.18	0.18
(3,3)	0.04	0.17	0.05	0.07	0.19
(4,3)	0.07	0.04	0.05	0.06	0.24
(5,3)	0.17	0.17	0.06	0.16	0.17

Table 4.3 The RMSEs between the predicted execution time values and the observed values during the simulation calculated by the network which is trained by trainbfg function.

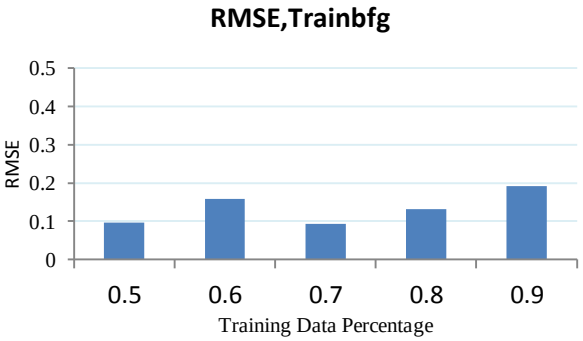

Figure 4.4 The RMSE values over the training data percentage during the simulation calculated by the network which is trained by trainbfg function.

4.4.3 Experimental Results of the Disparity Error Network

In this section, the neural network models the simulation configurations in order to predict the disparity error (disper). The neural network is trained by three desired training functions and the training data set is selected randomly with varying amount (50% to 90%). To compute the expected second output and RMSE value, the same DoEs should take into account with respect to different sets of known points (used as a training set).

Tables 4.4, 4.5 and 4.6, present the RMSE values that work on the disparity error (disper) and train by trainlm, trainrp and trainbfg functions respectively.

	Training data set Percentage				
(layers,neurons)	0.5	0.6	0.7	0.8	0.9
(3,1)	0.16	0.29	0.20	0.15	0.17
(3,2)	0.35	0.14	0.28	0.27	0.32
(3,3)	0.17	0.14	0.13	0.11	0.18
(3,4)	0.17	0.28	0.18	0.11	0.12
(3,5)	0.16	0.17	0.26	0.13	0.30
(1,3)	0.29	0.31	0.33	0.31	0.31
(2,3)	0.18	0.15	0.18	0.17	0.17
(3,3)	0.17	0.14	0.13	0.11	0.18
(4,3)	0.14	0.22	0.28	0.12	0.14
(5,3)	0.16	0.28	0.02	0.11	0.01

Table 4.4The RMSEs between the predicted disparity error values and the observed values during the simulation calculated by the network which is trained by trainlm function.

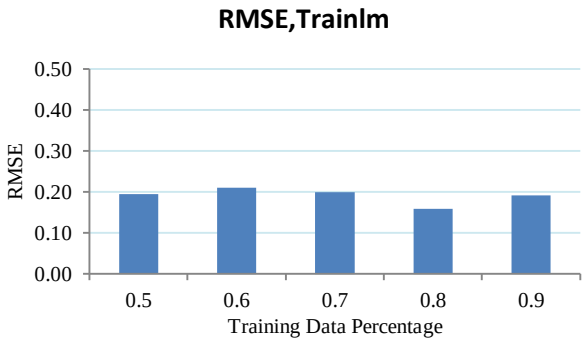

Figure 4.5 The RMSE values over the training data percentage during the simulation calculated by the network which is trained by trainlm function.

	Training data set Percentage				
(layers,neurons)	0.5	0.6	0.7	0.8	0.9
(3,1)	0.31	0.17	0.37	0.24	0.58
(3,2)	0.25	0.17	0.21	0.63	0.17
(3,3)	0.20	0.23	0.22	0.41	0.38
(3,4)	0.23	0.24	0.20	0.34	0.37
(3,5)	0.22	0.26	0.41	0.29	0.23
(1,3)	0.36	0.35	0.41	0.98	0.38
(2,3)	0.40	0.29	0.34	0.36	0.37
(3,3)	0.20	0.23	0.22	0.41	0.38
(4,3)	0.20	0.19	0.34	0.24	0.18
(5,3)	0.17	0.25	0.14	0.51	0.21

Table 4.5 The RMSEs between the predicted disparity error values and the observed values during the simulation calculated by the network which is trained by trainrp function.

Figure 4.6 The RMSE values over the training data percentage during the simulation calculated by the network which is trained by trainrp function.

	Training data set Percentage				
(layers,neurons)	0.5	0.6	0.7	0.8	0.9
(3,1)	0.14	0.14	0.13	0.19	0.30
(3,2)	0.20	0.19	0.33	0.32	0.27
(3,3)	0.17	0.19	0.20	0.36	0.41
(3,4)	0.23	0.30	0.25	0.20	0.20
(3,5)	0.20	0.20	0.14	0.34	0.34
(1,3)	0.29	0.32	0.32	0.31	0.32
(2,3)	0.33	0.27	0.22	0.18	0.23
(3,3)	0.17	0.19	0.20	0.36	0.41
(4,3)	0.17	0.15	0.21	0.31	0.19
(5,3)	0.18	0.15	0.15	0.21	0.20

Table 4.6The RMSEs between the predicted disparity error values and the observed values during the simulation calculated by the network which is trained by trainbfg function.

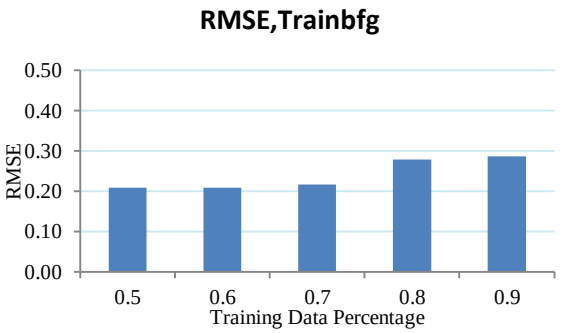

Figure 4.7 The RMSE values over the training data percentage during the simulation calculated by the network which is trained by trainbfg function.

For further information on the tables, bar charts which show the RMSE values over training data percentages are shown in figures 4.5, 4.6 and 4.7 as well.

Accordingly, in neural network, RMSE is one of the tools which lead to select the training function. The lower RMSE value means the better prediction of the output and, thus, higher performance in terms of accuracy. In figure 4.8, the bar chart on the left (a) depicts the RMSE values of the first neural network applied on first target output (execution time).

Each training data set percentage illustrates the differences of calculated RMSEs when different training functions have been employed. Based on this figure, training by trainlm function leads to lower RMSE while using the training data percentages of 50%, 80% and 90%. In other hand, trainrp and trainbfg cause the minimum RMS values when using 60% and 70% training data.

Figure 4.8(b), demonstrates the difference between RMSEs of the second output's (disparity error) predicted and true values. As it is shown in the bar chart, the minimum RMSE value in most of the training data percentages is devoted to trainlm function except for 60% training data in which trainbfg function overcomes it with less than 1% difference.

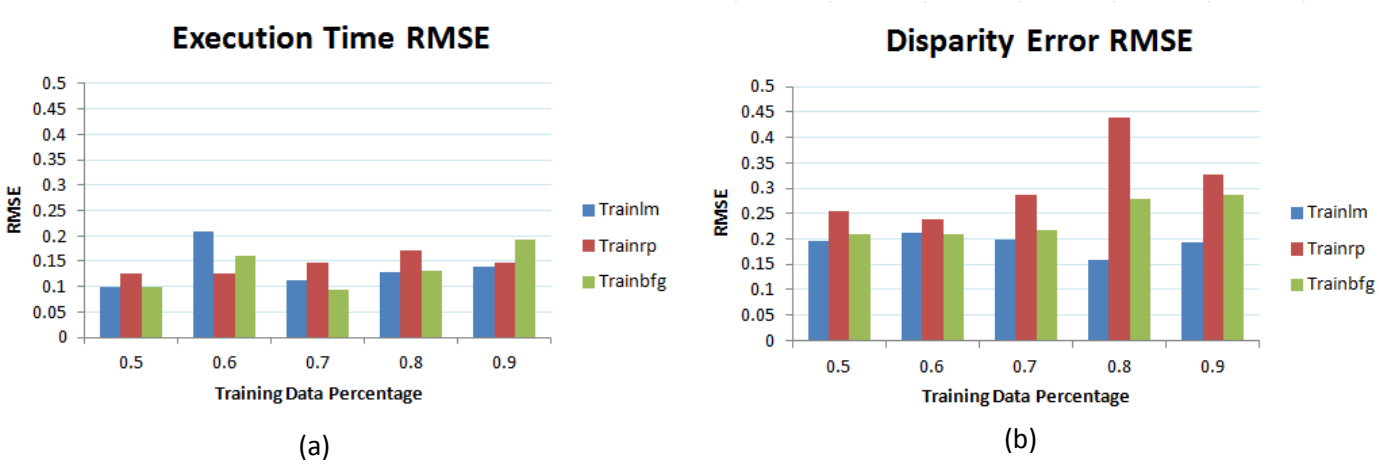

Figure 4.8 Comparison of the predicted and the expected output by different training functions of the neural networks predicting (a) execution time and (b) disparity error

Consequently, based on experimental results, Trainlm is the function which leads to the least RMSE, therefore it would be selected as the most efficient training function, with least prediction error.

The noticeable point of the two figures is the significant gap existing between RMSE values of the execution time on the left and disparity error on the right. It shows that execution time is much more predictable compares to disparity error. Furthermore, the difference of tools using for measurement of these two outputs could cause accuracy in prediction of RMSE. The execution time (wall-time) is measured by means of the P2012 with the posix-xp70 simulation model (based on ISS), while the disparity error (disper) is computed from each application.

In the next chapter, the methods of finding the most suited size of training dataset as one of the influential parameters in prediction of unknown data behavior will be explained.

4.5 Conclusions

The algorithm by which the neural network is trained is a parameter which has to be considered. The neural network tries to predict the execution time and disparity error of P2012. By comparing the predicted and observed values of these two outputs the RMSE values are calculated which are considered as a criterion to select the best training function. The minimum RMSEs are observed when we train by means of Trainlm function. Consequently, Trainlm is selected as the most efficient function with least prediction error and will be applied in the upcoming works.

CHAPTER 5

Design and Optimization of the Neural Network Model Using Cycle Accurate Simulations

Basically, the main goal of this study is to minimize the number of executed simulations during the exploration phase. Regarding this issue the number of simulated training samples should be as less as possible. The number of training samples is one of the complexity parameters of the DSE phase.

As we noticed, a cycle accurate system-level simulation is a long time process. Moreover in this type of simulation the exploration of the design alternatives would exceed the practical limits. To optimize the simulation of DSE, we needed to consider the minimum number of training data set with an acceptable amount of RMSE. To ascertain this goal, I have defined the training data set as 1% of the input data. In each of the iterations some more data would be adding up to the training samples set. This process would be continued until the training samples reach to 50% of the input data set. Accordingly, this procedure helps to find the minimum number of training samples as a percentage of the whole data set, to reach the RMSE below 20% (threshold value).

Regarding previous chapter, Trainlm has been chosen as the most fitted training algorithm to train the neural network. Thus, the training function employed to find the minimum training samples is limited to Trainlm.

The algorithm to find the minimum training samples, first starts with applying a certain number of neurons and layers to the Neural Network. The question of how many layers and how many hidden neurons should be there will be answered in this section. Second, the number of training samples is defined as 1% of the data set. Then, the Neural Network is trained by Trainlm training function with 10 different random extractions from the data. The RMSE between the predicted and observed training output will be calculated for each extraction. This process is iteratively repeated by increasing the training data set percentage up to 50% to derive a set of RMSE quantities for each training data set percentage.

Comparing RMSEs with a defined threshold (which is here 20%), the minimum number of training samples with its RMSE below the threshold will be found.

5.1 How many layers and Neurons

The question about the number of hidden layers and hidden neurons always comes up in any classification task of data using neural networks. Until today there has been no exact solution. Traditionally identification of topology has been based on trial and error and on pruning or constructive methods.

During the application of neural networks for the modeling, the same question always rises; and it is a critical question since the selection of topology has a profound impact on prediction results. In this thesis, we do not apply more than 5 layers and 5 neurons, rested on trial and error. Adding more layers and neurons increases the complexity of the topology. The proposed solution searches for the topologies, base on a novel fitness function, aiming to concurrently optimize performance while minimizing network complexity.

The assumption is made that compactness, i.e. having as few layers and neurons in the topology as possible, is considered an additional merit to overall verification set classification accuracy. It is essentially a multi-objective task of finding the most efficient and at the same time the less redundant network skeleton. As the optimal topology is judged by the capacity to generalize on unseen data, the most accurate structure will have fewer neurons than that suggested.

5.1.1 Design of Experiments

Basically, in terms of simulation, using statistical sampling techniques seems to represent the best solution to prune the different combinations of layer and neuron numbers. The sampling technique we employed is called Two Level Factorial Design. Each two factors of Factorial Design has two separate levels, thus the factorial experiment would have four different treatment combinations in total.

In statistics, the two leveled full factorial experiment is described as a design consists of two or more parameters. Each parameter has discrete possible values which let the experimental units take on all possible combinations of the minimum and maximum levels for such parameters. This experiment enables the evaluation of the effects of each parameter on the response variable, as well as the effects of interactions between parameters on the response variable [15].

Figure 5.1 shows the sample combinations which are selected by deploying Two Level Factorial Design. It is obvious that a set of 25 combinations is pruned to 4 treatment combinations- 1 layer and 1 neuron, 1 layer and 5 neurons, 5 layers and 1 neuron, 5 layers and 5 neurons. Applying this method leads to lessen the simulation time as well as decreasing the number of executed configurations to less than one fifth.

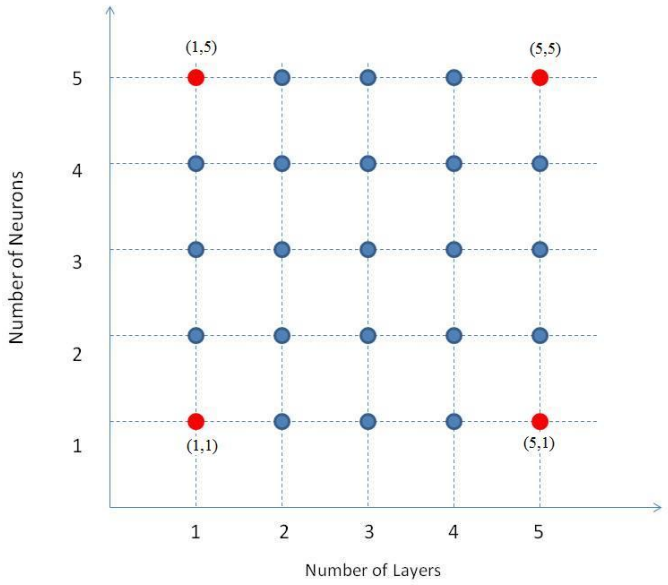

Figure 5.1 The red points show the 4 samples which have been selected out of the complete set of 25 samples by Two Level Factorial Design statistical sampling technique.

To obtain more accurate results, the flow of finding the RMSE, would be executed separately for two target outputs, wall-time (execution time) and disper (disparity error). The experiments' results are depicted by the box plots and line plots to create a coarse grain view of the RMSE over the training data set percentages.

5.2 Experimental Results of the First Network

As explained before, the first target output to be predicted by NN is wall-time which shows the execution time. In this section we work on four topologies which we have selected previously in the last part. Figure 5.2 shows the line plots of the average RMSEs in each training sample percentage for the 4 different treatment combinations.

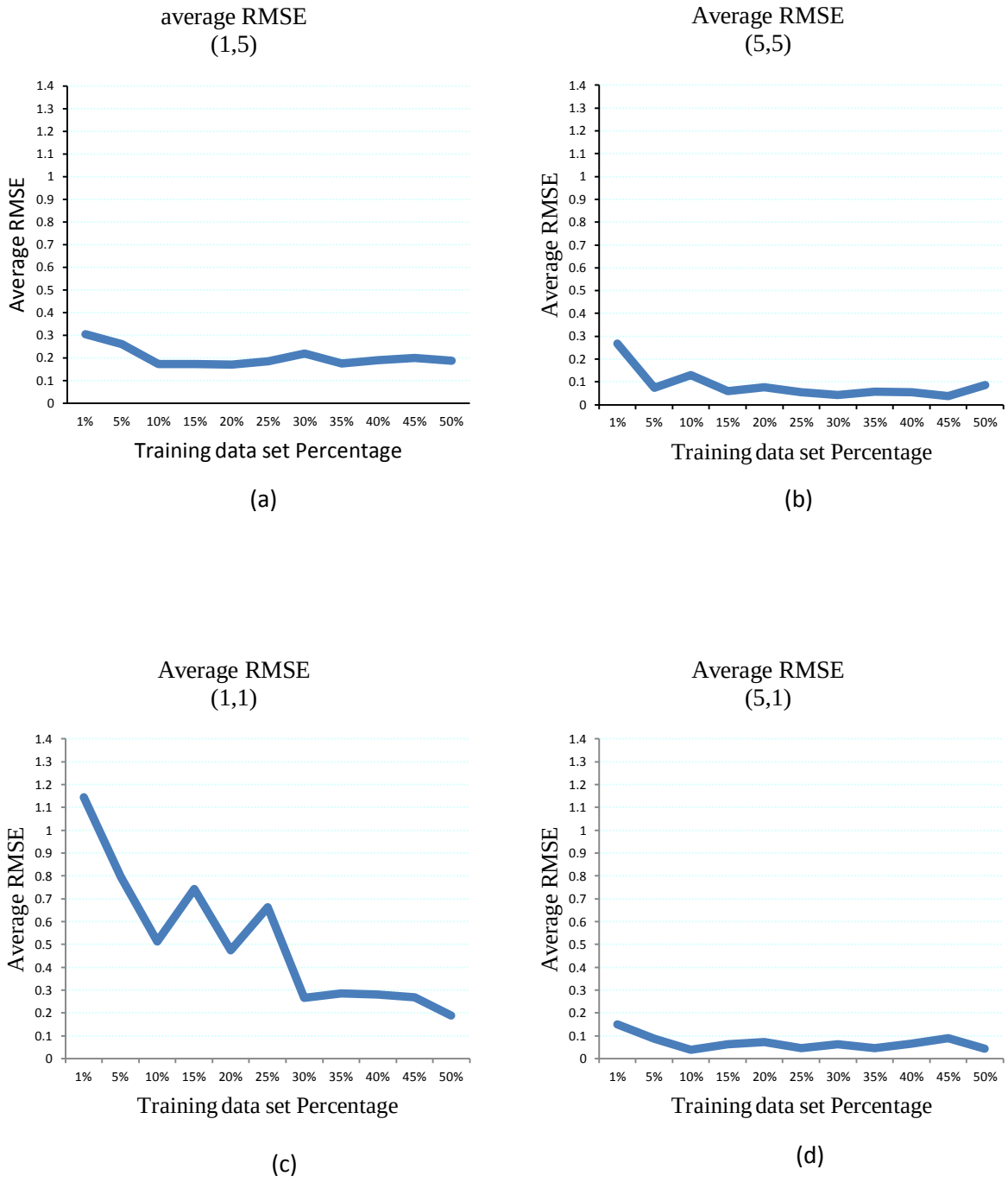

Figure 5.2 shows the average RMSE over training data set percentage for the first output, execution time, with (a) 1 layer and 1 neuron, (b) 1 layer and 5 neurons, (c) 5 layers and 1 neuron, (d) 5 layers and 5 neurons.

As it is shown in the line plots, the average RMSEs of the samples with 5 layers ((b) and (d)) are less than the other two samples with 1 layer ((a) and (c)). Although the RMSEs of these two samples are similar; But, as it is shown the network with 5 layers and 1 neuron has less fluctuation in comparison with the other one.

To select the optimal number of neurons between these two, we refer to the box plots showing execution time RMSE. Each box plot reveals the RMSEs which are collected from training with the same combinations of layer and neuron numbers. Based on the figure below, when we have 5 layers and 1 neuron (d), the error deviation is gradually decreased for the number of training samples more than 25% of the whole data set. The deviation of training samples between ranges of (10% to 25%) is negligible. On the contrary, the other network with 5 layers and 5 neurons, with 10% up to 25% of training samples, has a salient error deviation. Moreover, in the last box plot, the error deviation of NN when it works with 50% of training samples is remarkable. Regarding the average RMSEs that are demonstrated in line plots and the RMSE deviations which are shown in box plots for these 4 treatment combinations, the optimal combination which leads to minimum error between the predicted wall-time value and the observed value for wall-time, would be 5 layers and 1 neuron.

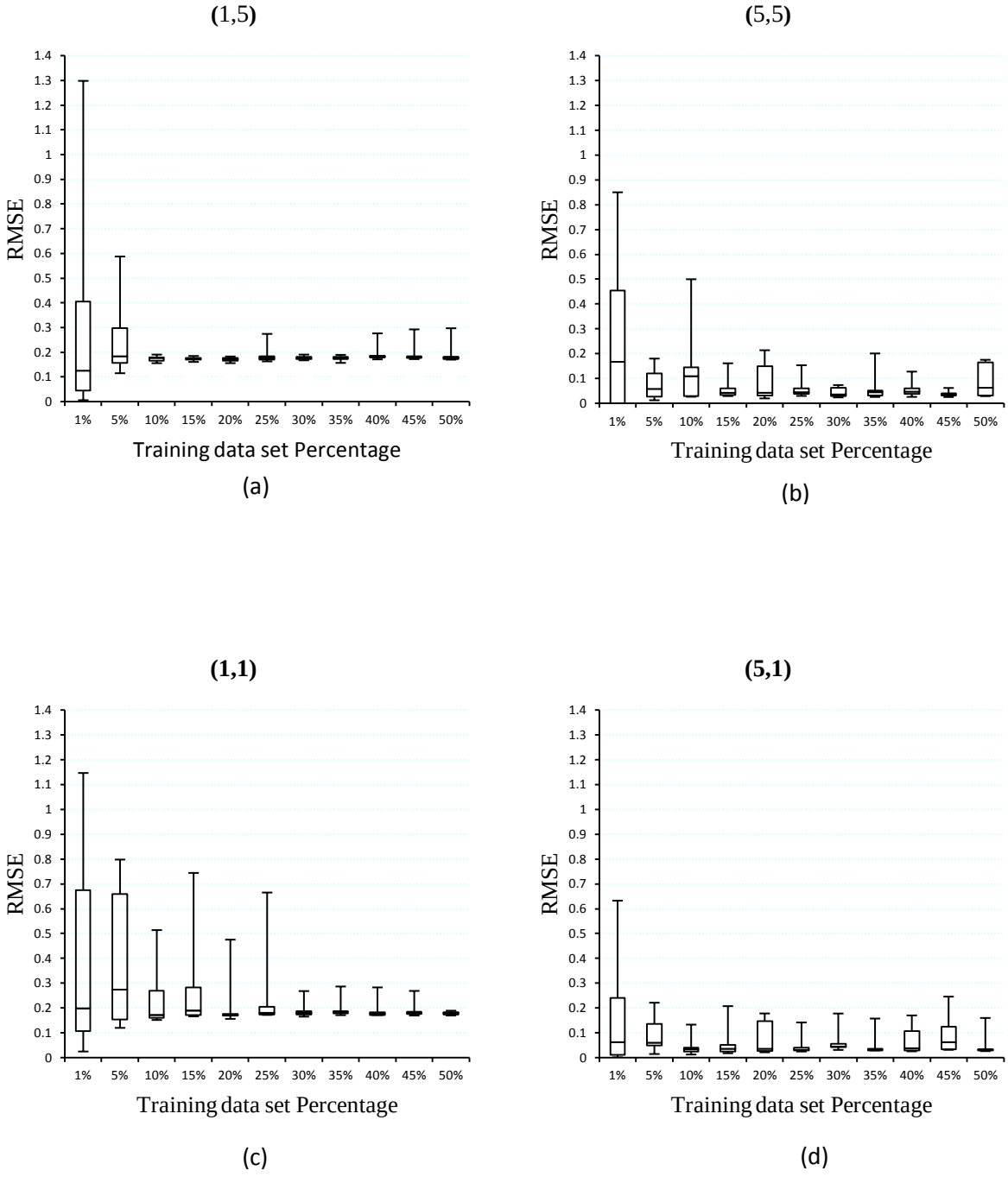

Figure 5.3 shows the RMSE over training data set percentage for the First Output, execution time, with (a) 1 layer and 1 neuron, (b) 1 layer and 5 neurons, (c) 5 layers and 1 neuron, (d) 5 layers and 5 neurons.

5.3 Experimental Results of the Second Network

The second neural network is modeled to predict disparity error which is called Disper. The same work flow as first neural network should be followed in order to select the most fitted combination of layers and neurons. Referring to the average RMSEs of the disparity error with 4 different topologies in terms of number of layers and neurons, as it is shown in the figure 5.4, it is gained that with increasing the number of layers the RMSE error decreases apparently. In first two line plots on the left, the RMSE is above 0.3 which is not acceptable comparing with the threshold. As it's shown below the average RMSE for NN with 5 layers decreases, however this decline is not stable.

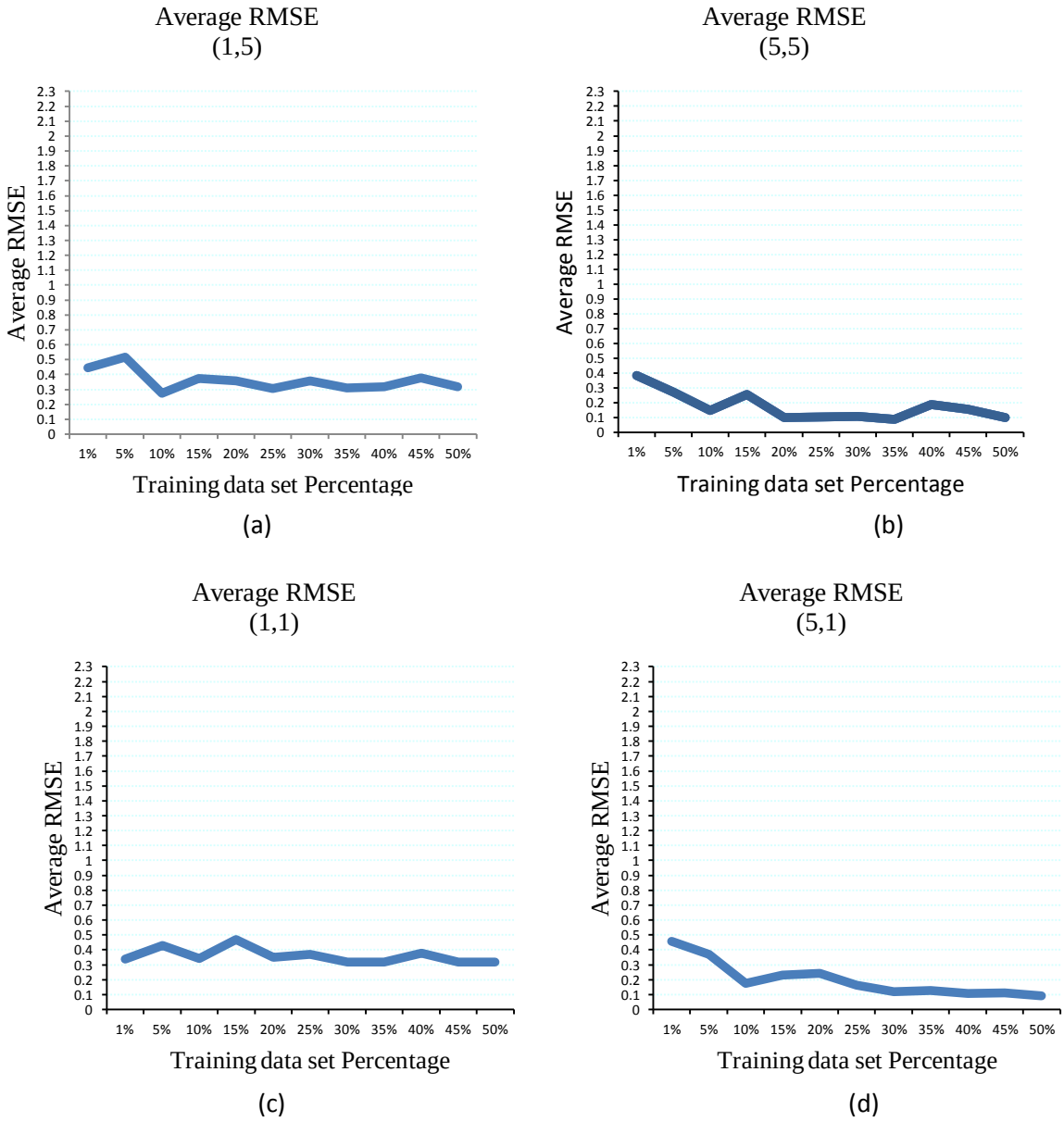

Figure 5.4 shows the average RMSE over training data set percentage for the second output (disparity error) with (a) 1 layer and 5 neurons, (b) 5 layers and 5 neurons, (c) 1 layer and 1 neuron, (d) 5 layers and 1 neuron.

Figure 5.5 illustrates the RMSE of the training data set percentage for disparity error of each treatment combination. In the right column of the figure, we can look at the RMSE of the five layered samples.

In figure (b) the RMSE of predicted disparity error between 20% and 35% of training data set percentage is 0.1 but it rises to 0.2 in 40% training data percentage. Despite the low quantity of RMSE for training samples above 20% in (b), figure (d) decreases gradually from 30% and becomes steady. The salient behavior of RMSE in (b) makes us to select the NN with 5 layers and 1 neuron. All together, the neural network which has 5 hidden layers and 1 neuron has the optimal output prediction RMSE for both execution time and disparity error outputs.

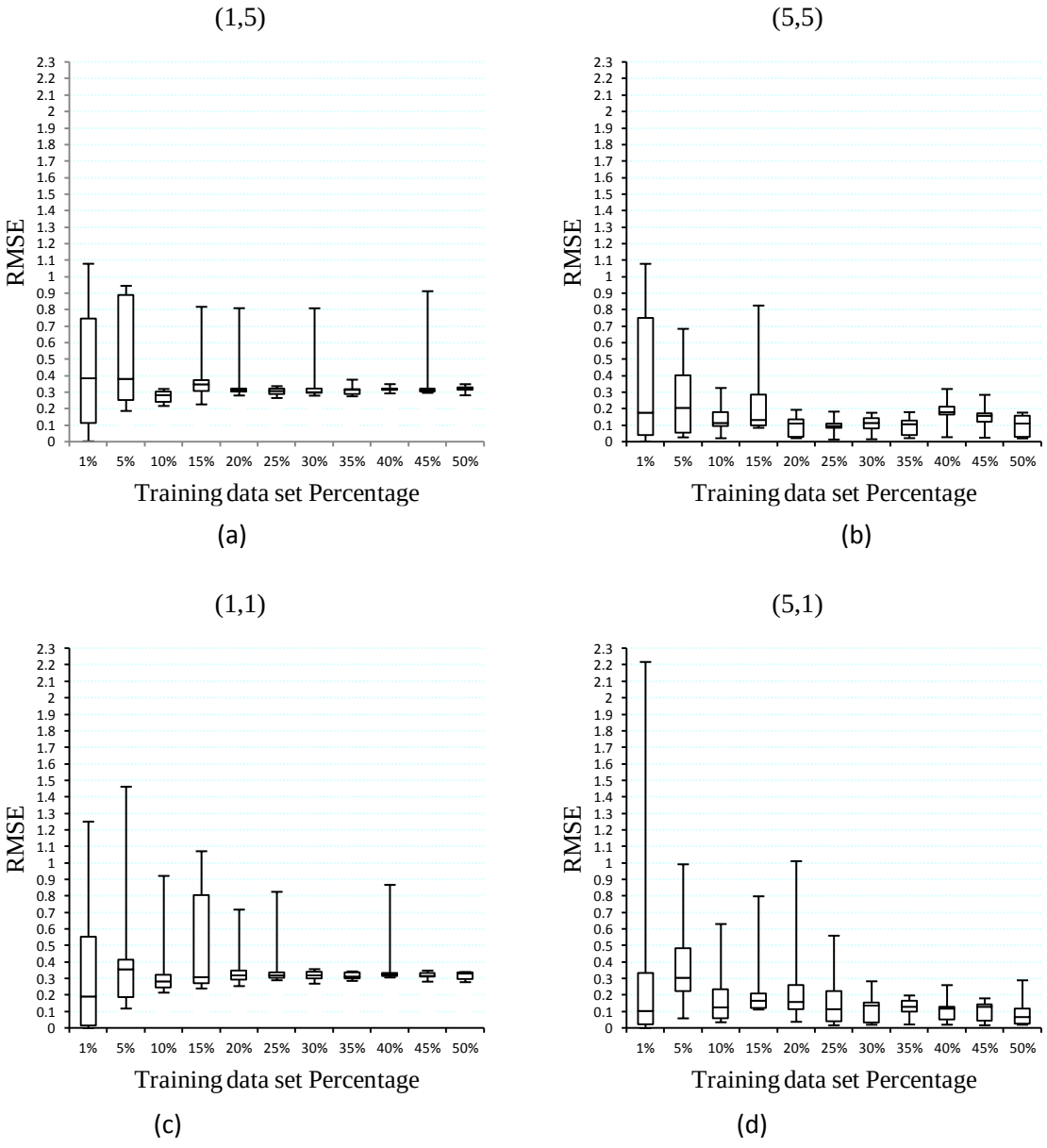

Figure 5.5 shows the RMSE over training data set percentage for the second output, Disparity error, with (a) 1 layer and 5 neurons, (b) 5 layers and 5 neurons, (c) 1 layer and 1 neuron, (d) 5 layers and 1 neuron.

Basically, the prediction of disparity error is more difficult than execution time. As it is illustrated in figure 5.6, the RMSE of execution time is pretty less than RMSE of disparity error. The two plots on the right side of figure 5.5, are not as steady as the average RMSE plots for the execution time prediction depicted in figure 5.3.

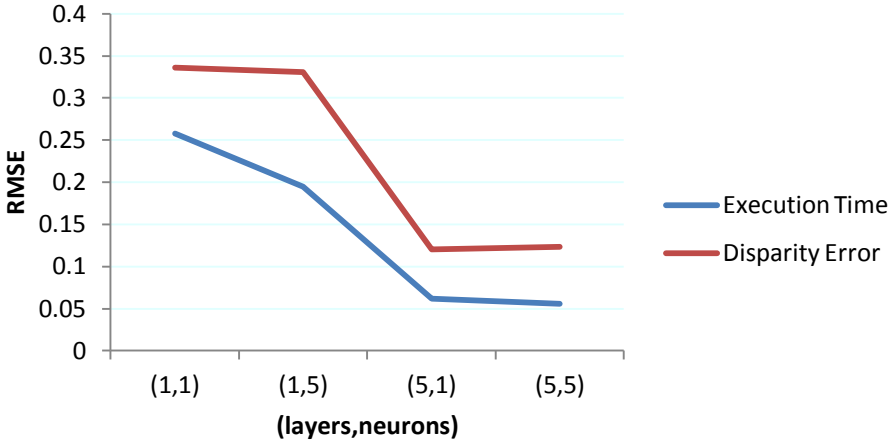

Figure 5.6 compares the prediction errors related to execution time network and disparity error network

In fact neural networks produce different results due to different initialization conditions even when everything else is kept fixed. This is why a single run is actually not enough to evaluate a topology. In this experiment, it has been found that the side-effects of the noisy fitness evaluation problem are reduced by increasing the number of samples. If, however, samples are available, the number of samples used should be progressively increased while observing the variance of RMSE results. There is a point where the inclusion of additional samples yields no benefit towards the stabilization of results.

Based on RMSE threshold which is set 0.2 in this experiment, the training sample sets which produce an error less than this value are acceptable.

It is understood from the figures that NN with 1 layer and 1 neuron is not sufficiently high-powered in order to predict the output. In the first NN which predicts execution time with a training data set more than 15% of the input data, all topologies except the first one with 1 layer and 1 neuron has acceptable result. The first topology can predict the output with RMSE equals to 0.2, when the training data set is above 30% of the input data. Second NN is the one which predicts disparity error, the two topologies with 5 layers can predict disparity error with RMSE less than 0.2, when at least 25% of the input data is allocated to training and the NNs with 1 layer are not able to attain a satisfactory prediction.

On one hand, adding layers to the topology improves the accuracy of the NN; on the other hand, adding neurons does not significantly improve the results but increases the complexity.

Obviously both complexity of the neural network and the value of prediction RMSE are remarkable to select the best model. The complexity of a neural network is the product of number of layers and number of neurons in each layer which means "number of layers * number of neurons ". As it is shown in table 5.1, product of complexity and RMSE can be used as a criterion to select the neural network topology. The value calculated for this parameter (shown in red) in a network with 5 layers and 1 neuron is lower than its value in a network with 5 layers and 5 neurons. In figure 5.7, the plots show the RMSE and complexity of training data set percentage for the network which is predicting execution time. Considerably, the topology with 5 layers and 1 neuron had the optimal results between all treatment combinations.

(layers,neurons)	(1,1)	(1,5)	(5,1)	(5,5)
average RMSE for training data set above 30%	0.26	0.19	0.06	0.06
complexity	1	5	5	25
complexity*RMSE	0.26	0.97	0.31	1.40

Table 5.1 The ratio of complexity and RMSE related to first neural network which predicts execution time

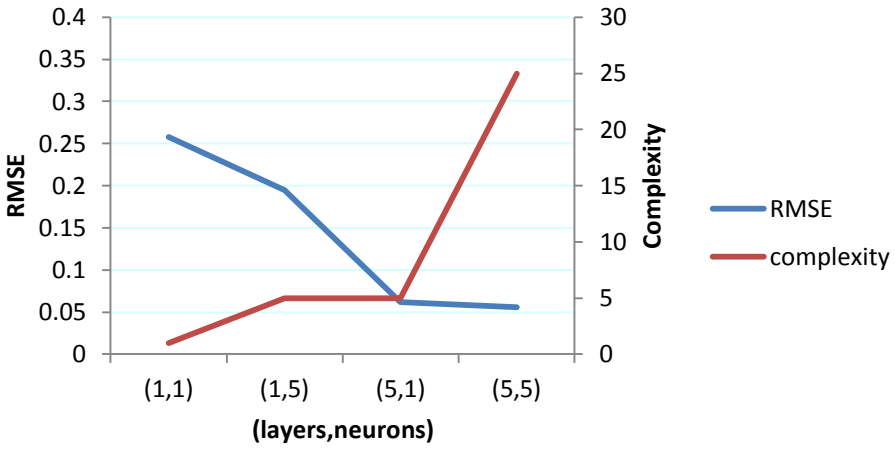

Figure 5.7 illustrates the line plots of complexity and RMSE related to the first neural network predicting execution time

Table 5.2, reveals the details of the network which predicts disparity error. Here also, the product of complexity and RMSE is used as a criterion to select the neural network topology. The values calculated for this parameter make the fourth row of the table. Neural network with 5 layers and 1 neuron has lower value (shown in red) compared to its value in neural network with 5 layers and 5 neurons. In figure 5.8, the plots show the RMSE and complexity over training data set percentage for the network which is predicting disparity error. In the second network also the topology with 5 layers and 1 neuron had the optimal results between all treatment combinations.

(layers,neurons)	(1,1)	(1,5)	(5,1)	(5,5)
average RMSE for training data set above 30%	0.34	0.33	0.12	0.12
complexity	1	5	5	25
complexity*RMSE	0.34	1.65	0.60	3.09

Table 5.2 The ratio of complexity and RMSE related to the second neural network which predicts disparity error

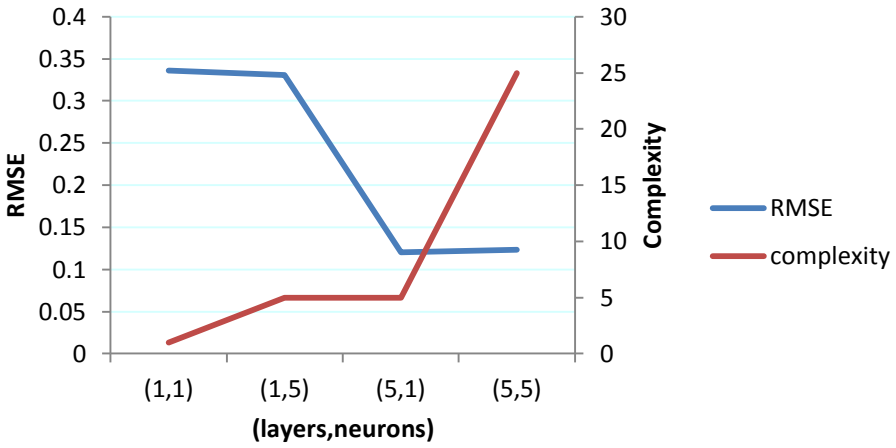

Figure 5.8 illustrates the line plots of complexity and RMSE related to the second network predicting disparity error

5.4 Conclusions

Putting into a nutshell, splitting samples to form a training data set with allocating at least 30% of the whole data set guarantees the neural network to predict the execution time with the RMSE below the threshold value (0.2). The disparity error neural network requires at least 25% of the input data to be used as training samples in order to predict with an acceptable RMSE.

Moreover, based on experimental results, the neural network topology consists of 5 layers and 1 neuron in each layer has the optimal result in both fronts, complexity of the network and RMSE between the visited output and the predicted output, for both neural networks predicting execution time and disparity error respectively.

CHAPTER 6

Using Native Simulations to Improve the Neural Network Model

As mentioned earlier, it takes half an hour to simulate a single configuration of the platform p2012 architecture. To decrease the simulation time further, in this chapter we introduce a multi level modeling methodology based on hybrid x86/posix-xp70 simulations.

Besides, we show the experimental results obtained by applying the proposed methodology to predict the execution time of the target configurations. Before moving through the details of modeling, simulating and analyzing the results, this chapter will give a clear illustration about the platform x86 and the overview of the project and the various steps taken towards accomplishing the goal and reaching results.

The platform x86 is a server machine with an AMD NUMA (Non-Uniform Memory Access) architecture with four nodes, where each node is a quadric-core processor. Therefore the tool has 16 processing-cores. Moreover, the machine runs Ubuntu Linux x86_64. The term x86 refers to a series of computer microprocessor instruction set architectures based on the Intel 8086 CPU. The x86 is much faster than posix-xp70 in simulating the configurations. Each configuration simulation consisting of 7 images takes in average 4 minutes which is one seventh times of the same simulation by posix-xp70. However, the results gained by x86 are not as accurate as the results gained by posix-xp70.

Using together posix-xp70 and x86 instead of using only posix-xp70 may decrease the time dedicated to simulate the training samples. The efficiency improvement can be investigated from two points of view; Either, with the same number of training samples, the time dedicated to simulating the training samples is decreasing; Or, with the constant time dedicated to configurations' simulation, achieving higher accuracy associated with numerous training data set is available.

6.1 The Structure of the Model

A design space as the input for the neural network is required. We collected the input data by simulating the configurations on posix-xp70 and x86. To provide a comprehensive validation, the analysis focuses on the architectural parameters listed in table 6.1, representing a design space composed of |X| =302 configurations to be simulated on posix-xp70 and a design space composed of |X| =3*18*6=324 configurations to be simulated on x86. The remaining 22 configurations are not feasible on the posix-xp70. The reason is that the memory available on the device is not enough to allocate the buffers required by these configurations. Regarding the application parameters, the table only consists of the list of parameters explored while the remaining parameters were kept constant.

The two DBs contain the exploration results obtained from the same design space which means that the same set of configuration parameters are used on two different platforms, the ISS simulator for P2012 (posix-xp70) and a multi-core x86 machine with native execution on Linux (no simulator).

Parameter	Possible Values
hypo_step	1-2-3
confidence	14-24-34-44-54-64
max_arm_length	1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18
match_limit	60
max_hypo_value	200
nb_wi_per_wg	16
nb_hypo_per_wg	2
nb_wta_workgroups	1
wg_col_pixels_width	80-96

Table 6.1 Input Parameters

The actual system response (represented in Table 6.2) consists of the average execution time (wall-time), energy consumption (local-mem-usage), disparity error (disper), CPU time (user-time) and number of pixels for each image (pixels).

Wall clock time is the actual time taken by a computer to complete a task. It is the sum of three terms: CPU time, I/O time, and the communication channel delay (e.g. if data are scattered on multiple machines). In contrast, CPU time measures only the time during which the processor is actively working on a certain task. Notably, wall_time and user_time for posix-xp70 are the same whereas for x86 they refer to different meanings. In this section, only the CPU time is taken into account which is independent from the other tasks running on the processor.

Local_mem_usage is the amount of local memory allocated for the algorithm. Disper is the average error per pixel. The result of the application is a disparity map, which is compared to a reference map (what we call the truth map) to calculate disparity error. It has to be considered that the total memory available on x86 is 32KB, versus 256KB on P2012) and the application metrics are calculated on 7 different test images.

Parameter	Description
Wall_Time	Execution Time
User_Time	CPU Time
Disper	Disparity Error
local_mem_usage	The memory used by Multi View in Bytes
pixels	number of pixels for each image

Table 6.2 The Output Parameters

6.2 The Modeling Methodology

In this section, the neural network models the configurations' simulation in order to predict the CPU time error (user_time). The neural network is trained by trainlm training algorithm and the training data set is selected randomly. The network topology is defined with 5 layers and 1 neuron. The same DoEs are taken into account with respect to different sets of known points (used as a training set) to predict the expected output and, eventually, to calculate RMSE values.

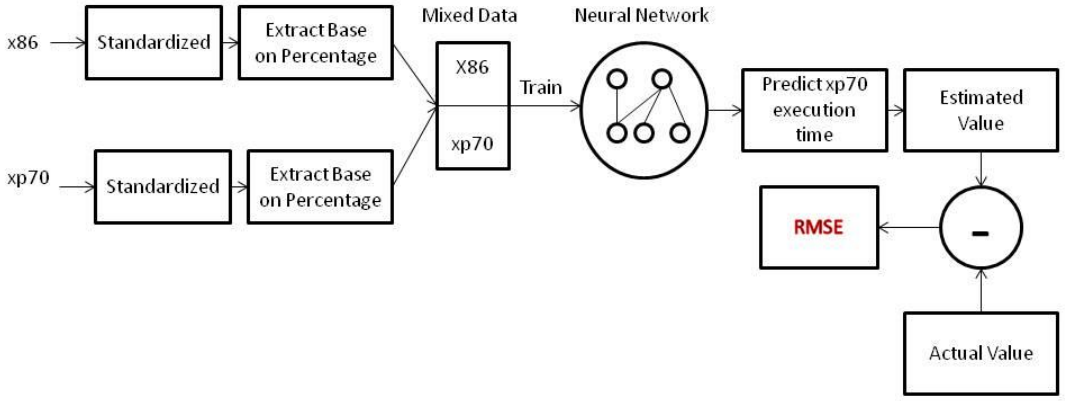

Figure 6.1 The flow of NN Modeling, using x86

6.2.1 Efficiency Improvement via Increasing the Accuracy

The maximum time dedicated to configurations' simulation is considered to be 5 hours. First, we train the neural network with configurations which are simulated on posix-xp70. The simulation process iterates when simulation time differs from 1 hour up to 5 hours. As each simulation takes half an hour to be completed on posix-xp70, it simulates 2 configurations in each hour. Hence, in the worst case (1 hour) the training data set consists of 2 samples while it contains 10 samples when the maximum time (5 hours) being dedicated to simulation.

We predict the CPU time which is equal to execution time in posix-xp70 and calculate the RMSE between the predicted values and observed values. As the data is gathered for 7 different images, we initialize the network 7 times and make the average RMSE of all the predicted user-times.

Next, we train the neural network with the mixture of configurations which are simulated by posix-xp70 and x86 together. The algorithm is to decrease the time dedicated to posix-xp70 for simulating the configurations and instead allocate it to x86. For instance when we have 4 hours of simulation by posix-xp70, we allocate 1 hour to x86 to simulate the configurations. Then we predict the CPU time and calculate the RMSE between the predicted values and observed values.

The experimental results of both neural networks are revealed in Table 6.3. In the first row, the time dedicated to simulation on posix-xp70 is shown in terms of percentage from 5 hours (maximum simulation time). In the second row, the calculated RMSE values between the predicted and observed CPU time when we simulate the training configurations by posix-xp70 are shown. In the third row, the values are related to RMSEs calculated by neural network which is trained by means of samples which are simulated by both posix-xp70 and x86. The last row shows the difference between the RMSE values calculated in both cases.

percentage of time from 5 hours dedicated to xp70	20%	40%	60%	80%	100%
xp70	1.40	2.98	1.70	1.20	1.33
xp70/x86	1.10	1.11	1.05	1.15	1.33
difference	0.30	1.87	0.65	0.05	0.00

Table 6.3 The RMSE values calculated by the neural network which is trained first, with posix-xp70 and second, with posix-xp70 and x86 together over percentage of time dedicated to xp70 to simulate the configurations.

Figure 6.2 illustrates the same information by means of line plots. The blue line shows the RMSE values between the predicted CPU time and the true CPU time when the neural network training samples are simulated by posix-xp70. On the contrary, the red line shows the same parameter when the configurations are simulated on posix-xp70 and x86. The lines do not behave in the same way. However they differ in quantity of the calculated RMSE values.

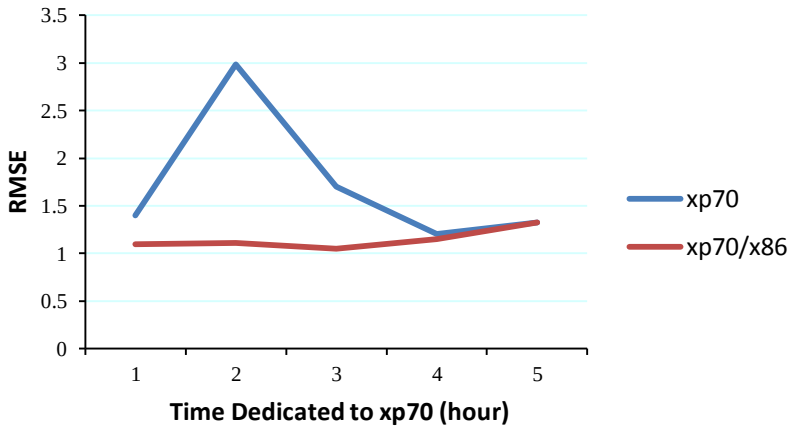

Figure 6.2 Line plots show the RMSEs over "Time dedicated to posix-xp70 " when simulations are done by posix-xp70 or posix-xp70/x86 together

By following the blue line, as it is expected, by reducing the time allocated to posix-xp70 to simulate the configurations, the number of training samples is decreased as well; And this consequently leads to the growth of the RMSE from 1.20 when 4 hours of the whole 5 hours are used to simulate the configurations up to 2.98 when just 2 hour is dedicated to posix-xp70 to simulate the configurations. Actually the value of RMSE is doubled by decreasing the time from 80% to 40%. Then by decreasing the simulation time from 40% to 20% the RMSE drops sharply down to 1.40 when just 1 hour is dedicated to simulation which means the network is trained with 2 training samples.

The red line follows the blue line when at least 4 hours is dedicated to posix-xp70 to simulate the configurations. By using x86 the number of training samples grows dramatically. Hence, the neural network trains with higher number of samples and predicts more accurate the user-time which leads to gain lower RMSE values. The quantity of reduction in RMSE values in each percentage of time dedicated to posix-xp70 is revealed in figure 6.3.

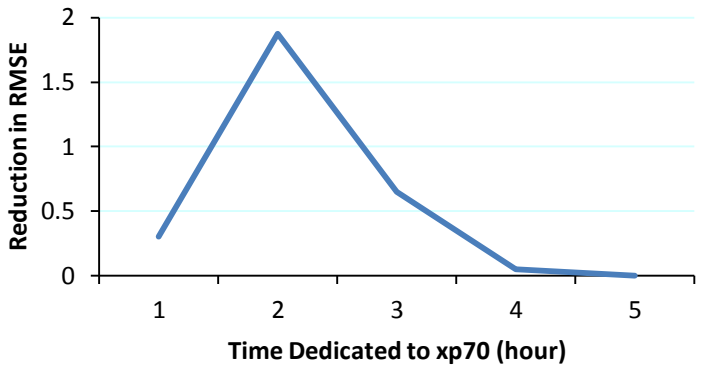

Figure 6.3 Line Plot shows the quantity of reduction in RMSE values by means of using x86 and posix-xp70 together to simulate the configurations.

As it is shown in figure 6.3, simulating configurations by means of posix-xp70 and x86 together improves the RMSE values at least 5% when 80% of the simulation time is dedicated to posix-xp70 and 20% is dedicated to x86. This improvement rises up to 180% when we allocate 40% of the time to posix-xp70 and 60% is allocated to x86.

When posix-xp70 simulates the configurations for 3 hours (60% of time), a RMSE equals to 1.70 is calculated. By allocating 2 hours more (40% of the simulation time) to simulation by means of x86, the RMSE value falls down to 1.05 which means 65% of reduction.

To conclude, using posix-xp70 and x86 together to simulate the configurations, improves the RMSE values. This improvement varies from 5% up to 180%.

6.2.2 Efficiency Improvement via Decreasing the Time

In this section, we trained the neural network by means of training samples which are simulated once with posix-xp70 and another time with posix-xp70 and x86 together in a way that 20% of the time is dedicated to posix-xp70 and 80% of the time is dedicated to x86. The goal is to find the speed up which is gained by allocating 80% of the simulation time to x86.

Figure 6.3 illustrates the reduction in RMSE values between predicted and observed user time when the configurations are simulated on posix-xp70 or posix-xp70 and x86 together. The blue line shows the prediction errors of the user time when the whole period of time is dedicated to posix-xp70 to simulate the configurations. The red line reveals the same parameter when 20% of the time is dedicated to posix-xp70

and the rest is dedicated to x86 for simulation. By allocating more time to simulation, the prediction accuracy rises in both cases. It means that when the number of samples used for training grows, the neural network can predict the target output better and thus, the RMSE between the predicted user time and the true user time values decreases consequently.

When the time is restricted, allocating more time to x86 to simulate the configurations improves the neural network predictions clearly with a big gap between the experimental results of RMSE when 100% of the time is dedicated to posix-xp70 and when 80% of the time is dedicated to x86 and only 20% is dedicated to posix-xp70 for simulation.

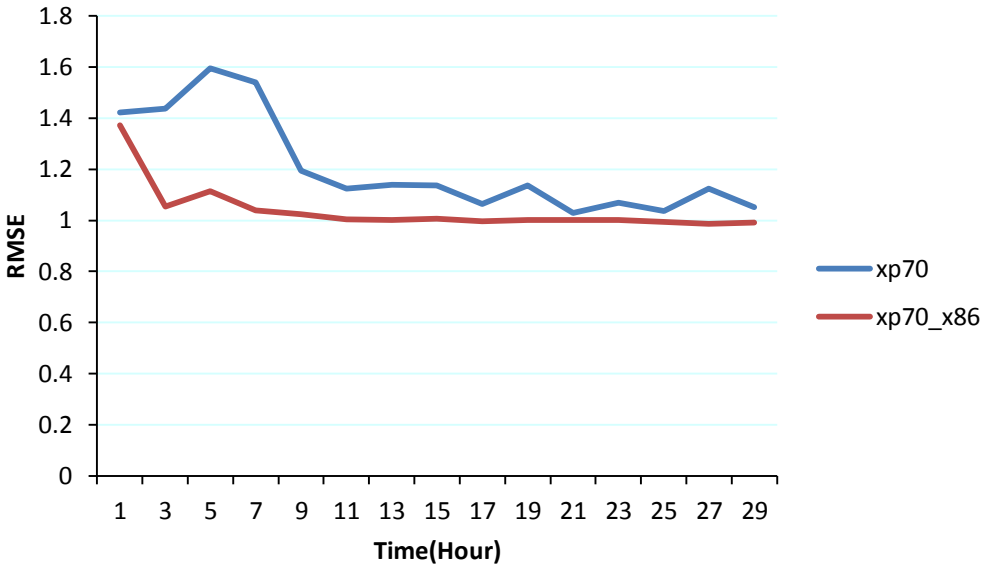

Figure 6.4 The RMSE values between predicted and observed user time over the time when the configurations are simulated on xp70 or xp70 and x86 together.

If we consider fixed values of RMSE, we can calculate the simulation time on posix-xp70 or on posix-xp70 and x86 together needed to achieve those RMSEs. The speedup value shows the division of the two calculated simulation time periods for each certain amount of RMSE.

The amount of speed up in terms of time needed to simulate the configurations between 7 to 19 hours. Dedicating 80% of the simulation time to x86 and 20% to posix-xp70 instead of allocating the whole simulation time to posix-xp70, can speed up the simulation time in gaining a certain RMSE.

Figure 6.5 illustrates the speedup values for the certain RMSEs achieved by using x86. As it is obvious in the line plot, the amount of speedup varies between 2 up to 8.

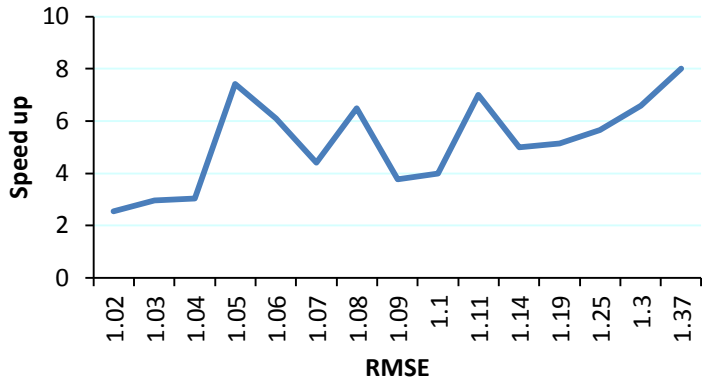

Figure 6.5 The speedup which is gained by simulating the configurations not only by posix-xp70 but also by means of x86 and posix-xp70 together.

6.3 Conclusions

We proposed a methodology of using hybrid x86 to simulate the configurations in order to improve the neural network model. The experimental results emphasize that simulating the configurations by means of posix-xp70 and x86 together does improve the neural network model in terms of time and accuracy.

On one hand, with allocating a certain amount of time to simulation, this methodology reduces the execution time prediction error from 5% up to 180% depending on the percentage of time dedicated to each simulator.

On the other hand, this methodology speeds up the simulation process between 2 to 8 to ascertain a constant execution time prediction error.

CHAPTER 7

CONCLUSIONS

In this dissertation, we have proposed a DSE methodology that leverages DoE paradigm and RSM techniques combined with a powerful way of considering customized application constraints. The DoE phase generates an initial plan of experiments in order to create a coarse view of the target design space; then, a set of response surface extraction techniques has been used to identify the non-feasible configurations and refine the Pareto configurations.

Artificial Neural Network as is proposed to address the problem of time consumption for simulation of an application targeted to Platform 2012.It's also tuned to build the response surface models and consequently speeding up the overall exploration phase. In fact, we propose a design space exploration methodology minding the statistical behavior of known configurations to predict the unknown configurations properties, by means of analyzing via simulations.

Using neural network with its advantages tackles the problem of DSE in two different ways. First, by reducing the time required to evaluate a system configuration and then by decreasing the number of configurations to be simulated.

In general, the proposed methodology (is capable of improving the accuracy and/or acceleration of configurations evaluation. The efficiency of this method depends on employing neural network parameters to be considered. A neural network with a topology consists of 5 layers and 1 neuron which is trained by Trainlm function is able to predict the execution time and disparity error of P2012 with an acceptable prediction error when at least 30% of the data set is allocated to training.

Moreover, the methodology of using hybrid x86 and posix-xp70 together to simulate the configurations improves the neural network model in terms of time and accuracy.

REFERENCES

[1] G. Ascia, V. C. (2007, october). Efficient design space exploration for application specific system-on-a-chip. J. Syst. Archit , 18.

[2] Givargis, T. V. (2002). System-level exploration for Pareto-optimal configurations in parameterized System on a chip. IEEE transaction on very large scale integration system , 416-422.

[3] Fornaciari, W. S. (2007). A sensitivity-based design space exploration methodology for embedded systems. Design Automation for Embedded , 7-33.

[4] Neema, S. S. (2002). Design-space construction and exploration in platform-based design, Tech. Institute for Software Integrated Systems Vanderbilt University Nashville Tennessee 37235 .

[5] S.G. Abraham, B.R. Rau, R. Schreiber, Fast design space exploration through validity and quality filtering of subsystem designs, Tech. Rep. HPL-2000-98, HP Laboratories Palo Alto (July 2000).

[6] R. Szymanek, F. Catthoor, K. Kuchcinski, Time–energy design space exploration for multi-layer memory architectures, in: Design, Automation and Test in Europe, 2004, pp.181–190.

[7] Hekstra, G. H. (1999). CPU64 design space exploration International Conference. International Conference on computer design, (pp. 599-606). Austin.

[8] Ghosh, A. &. (2004). Cache optimization for embedded processor cores: an analytical approach. ACM Transactions on Design Automation of Electronic Systems , 419-440.

[9] Eyerman, S. E. (2006). Efficient design space exploration of high performance embedded out-of order processors. DATE .

[10] Palermoa, G. S. (2005). Multi-objective design space exploration of embedded systems. Journal of Embedded Computing 1, 305-316.

[11] Joseph, P. J. (2006). A predictive performance model for superscalar processors. IEEE/ACM Int. Symp. Microarchitecture., , 161-170.

[12] Joseph, P. J. (2006). Construction and Use of Linear Regression Models for Processor Performance Analysis. In Proceedings of the 12th Inter International Symposium on High Performance Computer Architecture.

[13]Lee, B. C. (2006). Accurate and efficient regression modeling for microarchitectural performance and power prediction. Support Program. Lang. Oper. Syst. , 185-194.

[14]Ïpek, E. M. (2006). Efficiently exploring architectural design spaces via predictive modeling. Support Program. Lang. Oper. Syst.

[15]G. Palermo,C. Silvano, V. Zaccaria. A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration. IEEE transactions on Computer-Aided Design Of Integrated Circuits and Systems, Vol. 28, NO. 12, December 2009

[16]Palermo, G. S. (2008). An efficient design space exploration methodology for multiprocessor SoC architectures based on response surface methods . Proc. IC-SAMOS, (pp. 150-157).

[17]Silvano, G. P. (2008). An efficient design space exploration methodology for on-chip multiprocessors subject to application specific constraints.

[18]Luca Benini†, Eric Flamand, Didier Fuin, Diego Melpignano. Building an Ecosystem for a Scalable, Modular and high-efficiency Embedded Computing Accelerator. STMicroelectronics, Grenoble, France2012

[19]Ke Zhang, Jiangbo Lu, and Gauthier Lafruit. Cross-Based Local Stereo Matching Using Orthogonal Integral Images. IEEE Transactions on Circuits and Systems for Video Technology, Vol. 19, NO. 7, JULY 2009

[20]Ascia, G. C. (2001). Parameterized system design based on genetic algorithms . 9th international Symposium on Hardware/Software Co-Design, (pp. 177-182). Copenhagen.

[21]D. Pomerleau. Knowledge-based training of artificial neural networks for autonomous robot driving. In J. Connell and S. Mahadevan, editors, Robot Learning, pages 19–43. Kluwer Academic Press, Boston, 1993.

[22]Tesauro, G. (1995). Temporal difference learning and TD-Gammon. In Communications of the ACM , 58-68.

[23] Marzban., C. (2000). A neural network for tornado diagnosis. In Neural Computing and Applications (pp. 133-141).

[24] Mitchell., T. (1997). Machine Learning. Boston: WCB/McGraw Hill.

[25] Ananth Ranganathan.The Levenberg-Marquardt Algorithm. 8th June 2004

[26] Martin Riedmiller Heinrich Braun. A Direct Adaptive Method for Faster Back propagation Learning: The RPROP Algorithm. Institut fur Logik, Komplexitat und Deduktionssyteme University of Karlsruhe1993

[27] Rudy Setiono and Lucas Chi Kwong Hui. Use of a Quasi-Newton Method in a Feed forward Network Construction Algorithm. IEEE Transactions on Neural Networks,Vol. 6, NO. 1, JANUARY 1995

[28] T. J. Santner, B. Williams, and W. Notz, The Design and Analysis of Computer Experiments. New York: Springer-Verlag, 2003.