OPTIMIZED EMBEDDED ARCHITECTURES AND TECHNIQUES FOR MACHINE LEARNING ALGORITHMS FOR ON-CHIP AI ACCELERATION

by

SRIKANTH RAMADURGAM

B.E., Visvesvaraya Technological University, 2012

M.S., University of Southern California, 2015

A dissertation submitted to the Graduate Faculty of the

University of Colorado Colorado Springs

in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

2021

This dissertation for the Doctor of Philosophy degree by

Srikanth Ramadurgam

has been approved for the

Department of Electrical and Computer Engineering

by

Darshika G. Perera, Chair

Carlos A. Araujo

Terrance E. Boult

Thottam S. Kalkur

Byeong K. Lee

Date: February 12th, 2021

Ramadurgam, Srikanth (Ph.D., Engineering – Electrical Engineering)

Optimized Embedded Architectures and Techniques for Machine Learning Algorithms for On-Chip AI Acceleration

Dissertation directed by Assistant Professor Darshika G. Perera

ABSTRACT

In the era of smart and autonomous systems, machine learning is becoming the cornerstone of these systems. Machine learning, a subset of artificial intelligence, is being incorporated into various fields such as medical wearable in healthcare, smart cars in transportation, etc. Since most of these systems are typically realized on embedded devices, many machine learning applications are becoming common on these devices. However, embedded devices have many constraints: stringent area and power, increased speedup, reduced cost, and time-to-market. Furthermore, today's machine learning techniques are becoming increasingly complex requiring more processing power. Also, these systems require real-time processing and analysis, in order to make dynamic decisions. Consequently, new architectures and techniques are required to support and accelerate machine learning applications on resource-constrained embedded devices.

Neural Networks and Support Vector Machines are supervised learning algorithms that uses vast amount of data to train a machine learning model and draw inferences from the patterns. These algorithms incorporate statistical analysis and mathematical optimization techniques to analyze the data. Each data consists of large number of features to represent characteristics of a given sample. The ability to handle large-volume of data with high-dimensional features improves the accuracy performance of a machine learning model and is time-consuming and computationally expensive process. Currently, to accelerate the training of machine learning model, there are two popular hardware platforms, GPU and FPGA. GPU are easy to use and provide high

computing power. Due to their inherent parallelism, they provide significant acceleration compared to other platforms. However, GPUs are expensive and power-hungry devices. Due to high power consumption, the performance per watt cost reduces drastically. On the other hand, FPGAs provide low-power support with high parallelism and reprogrammable capabilities. But they provide limited computing power due to implementation complexity. Although, GPUs are widely popular in data centers, FPGAs can offer distinct advantages over GPUs with reprogrammable and reconfigurable abilities, which is most suitable for the evolving machine learning landscape.

Our goal in this research is to investigate and create customized embedded architectures to support on-chip hardware acceleration for data-intensive and compute-intensive machine learning applications.

In this work, our first contribution introduces embedded software and hardware architecture for support vector machines for both training and inference. We introduce novel and efficient system-level architecture for convex optimization to improve the accuracy performance of support vector machines. We introduce register transfer level details for different configurable kernels to handle linear and non-linear data for real-time applications. We architect efficient memory management techniques to build high-throughput and low-latency system-level design. The proposed architectures allow us to identify the benefits of internal structure of FPGA fabric to augment AI acceleration at low cost, low power.

The second contribution of this work introduces a design space exploration using parallel systolic arrays. Our contribution on design methodology and tradeoff analysis extends the adaptability of the platforms to a wide range of machine learning

applications. We elaborate on the limitations of system IPs and overcome the associated constraints of embedded devices.

The final contribution introduces embedded architecture for deep neural networks for supervised learning. We introduce the design methodology which illustrates the fundamental characteristics to take advantage of inherent parallelism nature of the deep neural networks. This design methodology also aims to improve the current platforms to develop a framework for ensemble architecture to cater for specific requirements of certain application with no additional cost. The architectures are generic, independent and configurable modules providing flexibility to enable real-time adaptable design to support AI acceleration. Besides being an important contribution by itself, this radical approach led to identify the benefits to establish a framework for future AI Chip design and on-device AI systems.

Based on the proposed hardware acceleration modeling, analysis and optimization of machine learning algorithms, this thesis offers a design methodology, guidelines, and experimental analysis for different machine learning applications. The experimental results and analysis presented herein demonstrate the feasibility of the research for resource constrained embedded systems and also can be further extended to different fields of applications in data science.

ACKNOWLEDGMENTS

A special thanks to my thesis advisor for the continuous support and encouragement throughout the coursework. The countless number of discussions I have had with Dr. Perera gave me an opportunity to work on the exciting research areas.

Also, I would like to express my sincere gratitude to the advisory committee members for their time and willingness to help me succeed. I greatly appreciate their support and guidance throughout the program.

A genuine gratitude and appreciation to all the outstanding members of ECE Department for giving me this opportunity. And finally, I would like to extend my best regards to all my colleagues and friends at the university.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AMBA	Advanced Microcontroller Bus Architecture
ASCII	American Standard Code for Information Interchange
ASIC	Application-Specific Integrated Circuit
AXI	Advanced Extensible Interface
BRAM	Block Random-Access Memory
CAGR	Compound Annual Growth Rate
CISC	Complex Instruction Set Computer
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Device
DDR	Double Data Rate
DMA	Direct Memory Access
DSP	Digital Signal Processor
EDK	Embedded Development Kit
FPGA	Field Programmable Gate Array
IDE	Integrated Development Environment
IPIC	Intellectual Property Interface Connection
IPIF	Intellectual Property Interface
ISA	Instruction Set Architecture
ISE	Integrated Synthesis Environment
MAC	Multiply-and-Accumulate
MIMD	Multiple Instruction Multiple Data
MMU	Memory Management Unit
PMIC	Power Management Integrated Circuit
RISC	Reduced Instruction Set Computer
SDK	Software Development Kit
SIMD	Single Instruction Multiple Data
SMO	Sequential Minimal Optimization
SRAM	Static Random-Access Memory
XPS	Xilinx Platform Studio

CHAPTER 1

INTRODUCTION

1.1 Introduction and motivation

Embedded devices have significantly evolved over the past few years playing a key role in the overall digital transformation. This includes rapid adoption of embedded devices to build smart and connected ecosystems. At the same time, it has led to major research interest to develop complex and efficient embedded design supporting wide range of applications. On the other hand, embedded devices have numerous challenging factors including stringent area, low cost, power consumption, and limited memory. Consequently, these constraints pose serious challenges to the embedded systems designers.

According to global statistical analysis [1], the market share for embedded hardware devices is projected to grow at 6.1% compound annual growth rate (CAGR) from 86.5 billion (USD) in 2020 to 116.2 billion (USD) by 2025, compared to embedded software forecast [2], 12 billion USD in 2019 to 20 billion USD by 2025. The promising growth rate of embedded hardware is expected to cover majority of the market share for various application domains.

One such application domain that has become very common on embedded devices is machine learning. Machine learning is a statistical method to automate data analysis and is a subset of Artificial Intelligence. Innovations in artificial intelligence are the key driving factors for developing powerful machine learning techniques. Some of the applications of machine learning are: computer vision, medical diagnosis, online customer support, speech recognition etc.

Increasingly, machine learning plays a crucial role in our day-to-day activities. However, machine learning has many constraints in the real-world applications. One such constraint is the speed performance. Machine learning involves two tasks: training and testing (inference). During the training process, the machine learning model processes the input data to extract important features. During the inference, the extracted features are used to make output prediction. With the increase in the amount of data used for training, the computational time increases exponentially. This exponential increase in the computing time hinders the speed performance.

In order to satisfy the speed constraints, various platforms are used to improve the speed performance such as general purpose dedicated Application-Specific-Integrated-Circuits (ASICs) (including AI (Artificial Intelligence) chip), Graphical Processing Units (GPUs), and Field Programmable Gate Arrays (FPGAs). A general purpose processor provides high flexibility to support wide of range machine learning applications. However, the processing speed is unable to keep pace with the computational demands of machine learning applications. ASIC's are one of the power efficient options and fabricated for a specific application. Although, the ASICs is a feasible option to provide hardware support to improve speed performance, changing the architecture and fabricating a new integrated circuit is a high risk and time consuming process. On the other hand, GPUs are the popular platforms for machine learning and provide immense amount of processing power to accelerate the machine learning applications, but they are power hungry devices diminishing overall performance per watt performance. Comparatively, use of FPGAs is dramatically increasing due to their flexibility to support

various machine learning applications, are significantly faster and one of the power efficient platforms.

In 2010, Microsoft research group [3] proposed augmenting the Central Processing Units (CPUs) with FPGAs to enable machine learning applications. Eventually in 2016, Microsoft incorporated FPGAs for inference. However, despite the high cost and high power consumption, GPUs dominate the training process of machine learning. Although, GPUs provide a potential solution to solve the speed performance issue of compute-intensive and data-intensive machine learning applications, the power consumption is significantly higher. [4] illustrates the power consumption of GPUs are almost 45x times that the FPGAs. The high-power consumption reduces the overall performance per watt and is not a viable option for portable applications.

Our previous analyses [131],[202] illustrates that FPGA-based systems are currently the best avenue to support compute and data-intensive applications and algorithms running on resource-constrained embedded devices. Furthermore, our previous work on FPGA-based accelerators, architectures, and techniques for various compute/data-intensive applications, including data mining and data analytics [127],[128],[203],[204],[205],[206],[207], control systems [208],[209],[210], security [211],[212],[213], machine learning [214],[215],[216],[217], and bioinformatics [218], demonstrates that FPGA-based systems are the best option to support and accelerate complex algorithms on embedded devices, considering the constraint associated with these devices and the requirements of the applications running on these devices.

In this research work, we introduce customized and optimized embedded hardware architectures to support and accelerate machine learning applications for

portable embedded devices. We utilize the FPGA platforms to illustrate the feasibility, efficiency, and scalability of the platform for data-intensive and compute-intensive applications. We also introduce systolic array architecture to demonstrate the crucial architectural tradeoff to support and accelerate machine learning applications. In order to achieve our goals, we have formulated our research objectives as mentioned in the following section.

1.2 Our research objectives

The main objective of our research is to provide customized and optimized embedded architectures to support on-chip hardware acceleration for machine learning applications.

In order to achieve our main research objective, we have divided our research work into three major stages. The objective of each progressive stage is as follows:

- In the first stage, we explore the scalability, efficiency, and feasibility of the embedded hardware architectures to support both training and inference for machine learning algorithms specifically for support vector machines and deep neural network.
- In the second stage, we investigate the potential gain and the trade-offs with respect to parallel processing, speed-performance, area-efficiency, and power analysis of hardware architecture for the aforementioned machine learning algorithms.
- In the third stage, we analyze and develop the design methodology to build system-level architecture for FPGA-based design, considering the associated constraints of embedded platforms. Formulating the design methodology includes

trade-off analysis for area and execution time (thus, speedup), number of instances for parallel systolic array, adaptability of hardware design for different machine learning approaches, efficient memory management techniques etc. We aim to incorporate the insights gained from the two previous stages to develop efficient hardware architecture for Convolutional neural networks.

1.3 Our contributions

In this research work, we make three major contributions corresponding to the three major stages mentioned above.

For the first stage, we introduce optimized FPGA-based embedded hardware architecture for support vector machines. We introduce embedded software and hardware architecture to adapt an algorithm in different platforms. We design and integrate pre-fetching and burst transfer techniques to reduce the memory access latency and to facilitate real-time processing. We provide different configurable option for mathematical kernels, which enables the user to select and utilize the most suitable kernel for linearly and non-linearly separable data. We also introduce a generic chip-level hardware support for convex optimization and achieved substantial performance gain to process large-volume of data.

Our second major contribution is the design space exploration using parallel systolic arrays. We introduce efficient and unique embedded hardware architecture with systolic array configuration to accelerate both training and inference for data classification. Based on the experimental analysis, we illustrate architectural tradeoffs to distinguish the critical and main contributing factors for hardware acceleration. We

address and overcome the associated constraints of embedded platforms such as limitations of hardware IPs capabilities and incorporating the IP modules for SoC design.

For the third major contribution, we utilize the outcome from stage 1 and stage 2 to build efficient neural network hardware architecture to support on-chip AI acceleration. We introduce the design methodology which illustrates the fundamental characteristics to take advantage of inherent parallelism nature of the deep neural networks. This design methodology also aims to improve the current platforms to develop a framework for ensemble architecture to cater for specific requirements of certain application with no additional cost.

1.4 Dissertation organization

The Dissertation is organized as follows:

In this chapter, we have presented a brief overview to introduce embedded platforms, machine learning, our objectives and motivations for our research work.

In chapter 2, we present the background study for different means of hardware support including general-purpose processor, ASICs, GPUs & FPGAs. We discuss the framework for hardware and software architecture. We present the details of existing works on hardware implementations for similar machine learning algorithms.

In chapter 3, we present our design approach, system-level architecture, and novel techniques to support machine learning applications are detailed in this chapter. Our novel, unique, customized and optimized FPGA-based embedded hardware accelerator for three stages of CO-based SVM algorithm is introduced along with our embedded software design.

In chapter 4, we present background study for systolic arrays, description for the design approach and framework of parallel systolic configuration for hardware accelerator. Experimental results and analysis are reported. Experiments are carried out to illustrate the feasibility and scalability of the embedded design for machine learning.

In chapter 5, we present the architectural details for deep neural networks and discuss the potential of ensemble architecture for different applications. Discussions for experimental results and analysis in terms of timing, speedup, area, and accuracy are reported.

In chapter 6, we summarize our contributions with concluding remarks and future directions for our research work.

CHAPTER 2

BACKGROUND STUDY

In this chapter, we present the background study for our research. Recently, machine learning applications are commonly implemented on embedded devices because of low cost, low power, portable, high performance etc. [5]. In section 2.1, we discuss various means of computing platforms for application specific operations. In sections 2.2, we elaborate on the framework for machine learning applications and techniques. Also, we discuss the optimization techniques utilized by the machine learning algorithms to improve the reliability of the machine learning applications. We provide the details about the existing work on hardware support in this section.

In this chapter, we provide the necessary mathematical equations to illustrate the development of classification algorithm; however, for extensive details about the derivation of these equations, data visualization we refer to the related materials in the existing literature. Few recommended pre-requisites include fundamental concepts of digital design flow, logic synthesis, RTL design for hardware architecture, probability theory for machine learning, linear algebra and calculus.

2.1 Hardware Platforms

In this section, we discuss about the various means of computing platforms suitable for different application domains. Some of the most commonly used hardware platforms are general-purpose processor, application-specific integrated circuits, graphic processing units and reconfigurable devices [6]. The anticipated new-emerging technologies such as quantum computing, photonic computing and other exascale

computing projects are beyond the scope of this research work. In the following subsections, we discuss brief overview of each of them.

2.1.1 Microprocessors

The general-purpose micro-processors are the most predominant computing platforms widely used in all the desktop computers, laptops etc. Microprocessors are designed to execute specific set of arithmetic & logical operations and provide high flexibility to develop large number of applications [7]. In the past few decades, significant research work has been devoted to improve the performance by incorporating different techniques and methodologies.

The flexibility offered by a microprocessor comes at the cost of inferior performance for specific applications. Power consumption of these devices is much higher and is not suitable for specific applications. Power consumption is a critical issue for portable and embedded devices [8]. In order to address this issue, Application Specific Integrated Circuits (ASIC) plays a major role to overcome the power issue, including others such as area. We discuss the details of the ASICs in the following sub-section.

2.1.2 Application Specific Integrated Circuits (ASIC)

Unlike microprocessor, ASIC are dedicated hardware module for a particular use. Application-Specific Integrated Circuits (ASIC) as the name suggests it is application specific IC targeted to perform a specific task [9]. Since these integrated circuits are customized to perform only a specific task, these devices are small, fast, power efficient systems with minimal routing & timing issues. However, the applications cannot be modified once after the device is fabricated compromising the flexibility criteria. In case,

any additional functionality needs to be added to the existing chip design, it requires re-fabrication, which in turn a time consuming process. Many companies have developed their own AI chips to support and accelerate the current global demand for AI applications.

2.1.3 Graphics Processing Unit (GPU)

Graphic processing Units (GPU) are specialized devices to accelerate and perform mathematical computations primarily for graphic display devices. Recently, there is a growing interest to utilize GPU for non-graphic applications [10] [11]. GPUs have thousands of cores per chip favorable for high-performance computations with high memory bandwidth. Most of the applications implemented on a CPU will most likely utilize the GPU as a co-processor design to accelerate an application. These devices have shown significant improvement for applications ranging from graphic displays to scientific computing. Although GPUs provide high computing power but they are expensive devices with high power consumption. In order to utilize the full potential of a GPU requires a great deal of efforts from the user/designer.

2.1.4 Reconfigurable computing platforms

Field programmable Gate Arrays (FPGAs) are one of the most popular reconfigurable devices [12] [13]. Due to their inherent nature of re-programmability, a number of different applications can be efficiently developed and prototyped for faster time-to-market requirements. FPGAs also offer other useful solutions to develop complex design such as providing embedded processor, DSP blocks, ready-to-use IP cores, custom IP cores, I/O interfaces etc [14]. The important factors such as low-power requirements, less system cost, adaptable platforms provides an opportunities to design a high-

performance computing platforms for large-scale data applications such as machine learning. However, the run-time to translate and map an RTL design takes significant amount of time and in certain cases routing the individual blocks fails due to limited interconnect resources. In such cases, the user needs to carefully modify the design to meet the synthesis rules.

2.2 Machine learning

Machine Learning (ML) is one of the most important research areas and is continuously evolving in many different fields. Some of the interesting areas of applications are autonomous cars, biometrics, computer vision, geo-statistics, medical diagnosis, speech recognition, automated trading platforms, hand writing recognition, virtual personal assistant etc. [15]. According to a market research [16], the global market for machine learning was $1.4 billion in 2017 and is expected to reach $8.8 billion by 2022 with an annual growth rate of 43.6%. The aforementioned facts demonstrate that machine learning market will continue to thrive as smart and autonomous systems emerge.

Machine learning, a subset of artificial intelligence which enables a system to learn, identify patterns, and make decisions without being explicitly programmed, and with minimal or no human intervention [15]. In these fields, the system should be capable of making critical decisions based on primary characteristic features. Machine learning typically involves many important data mining tasks, and can be categorized into supervised learning (i.e., classification) and unsupervised learning (i.e., clustering) [15]. The basic premise of machine learning is to create techniques that can learn from the sample input data (known as the training data), by performing statistical analysis, in order

to make accurate predictions or decisions on the data [17]. With the advent of smart and autonomous systems, machine learning is becoming the cornerstone in creating these systems, in which an autonomous car making a decision to determine the type of object in front. Also, in the field of medical diagnosis, these decisions have helped many doctors to diagnose whether a certain tumor is malignant or benign.

In the following sub-sections, we briefly discuss the necessary details required for illustrating our hardware implementation in section 3. Although, we provide the necessary mathematical equations to describe certain concepts of the classification and mathematical optimization, we also direct to specific proceedings for extensive details.

2.2.1 Classification of Machine learning

Developing a machine learning models require a specific set of data called training set and testing set. Each set of data samples consists of an input vectors, denoted by x. Each input vector, x is composed of N number of features representing a specific characteristic of the associated datasets called training set, denoted by $\{x_1, x_2, x_3, \ldots, x_N\}$. Each training set is systematically categorized into respective output labels called target value, denoted by y_t. On the other hand, testing sets includes only the input vectors and the machine learning will predict the associated target values based on the features learned during the training process. The performance of the machine learning algorithms is evaluated based on the ability of machine learning models to categorize correctly; this is known as generalization [15].

Based on the utilization of the datasets, machine learning is broadly classified into supervised learning, unsupervised learning and reinforcement learning. Each learning process involves training and testing stages. From [18], the four popular subclass

involved in machine learning process are clustering, Classification, Regression and Dimensionality Reduction. Clustering (unsupervised learning) [19] and Classification (supervised learning) are most commonly used and plays a crucial role to determine the category of an input sample. A regression method predicts the outcome of an input data based on certain attributes and dimensionality reduction technique helps us to distinguish between the non-trivial feature vectors. For our research purpose, we will limit our focus to classification methods.

In supervised learning, the machine learning models is trained using a set of pre-defined input vectors and output labels (training set). If the output vectors are to be evaluated and assigned to a finite number of categories, such a process is called as logistic regression, also referred as classification. Examples for classification are handwriting analysis [20], cancer diagnosis [21], predicting stock prices [18] etc. On the other hand, if the output values involves more than one continuous variables, such a process is knows a linear regression. Examples for linear regression include real-estate price analysis, melting of polar ice caps etc.

In unsupervised learning, the training set consists of only the input vectors without the corresponding output labels [15]. In this process, a model groups similar set of data to predict future outcomes, this is called as clustering. Examples for unsupervised learning include grouping customers based on their purchasing habits, grouping astronomical objects etc. In the unsupervised learning, an association rule is also often used to discover patterns that describe large portions of the available data sets, for examples, customers in a specific group also tends to buy from other facet. Association based learning is popular in the area of retail and logistics.

In the reinforcement learning, the objective is to maximize the rewards by identifying a suitable action for a given situation [15]. The machine learning models learns based on multiple attempts to learn from trail-and-error basis. For example, reinforcement involves training a computer to play chess by thousands of trails and exploring possible suitable action for a given situation.

In this research work, we focus on supervised learning (specifically, classification) techniques. Classification techniques are employed by many machine learning applications in various domains including medical, finance, and transportation. There exist many classification algorithms; hence, selecting a suitable algorithm for a specific application (or dataset) would significantly impact the accuracy of the results as well as the overall system's performance [15].

For instance, employing a linear classifier for non-linear separable data can diminish the accuracy and efficiency of the results, which in turn can hinder the subsequent analysis [15]. As a result, we investigate different classifiers (or classification techniques/algorithms) utilized for machine learning applications. Next, as a case study, we decided to focus on the Support Vector Machine (SVM) classification algorithm [22] [23] [24] [25], since SVM has many advantages/traits that are deemed suitable for machine learning applications.

For instance, SVM is more appropriate for classifying non-linear separable data [26], enables classification in multi-dimensional space [24], is capable of handling high dimensional data [27], and generate high accuracy results [27], [24]. In addition, the accuracy and classification process of the SVM can be further improved by incorporating non-linear optimization methods to find the optimal solutions [27]. Hence, we decide to

integrate the convex optimization (a widely used non-linear optimization methods) to the SVM classifier in order to clearly distinguish between the two separate classes by maximizing the margin width of the hyper-plane, which in turn leads to an optimal solution [27].

2.3 Support Vector Machines (SVM)

The support vector machine (SVM), first introduced by Cortes and Vapnik in 1995 [28], is one of the novel machine learning techniques, based on the statistical learning theory [29]. The SVM was initially developed for classification tasks, and was later extended to regression analysis [30]. As stated in [15], [31], the SVM classifies the data points based on its location with respect to the hyper-plane. We investigate the published literature to get an insight into the concepts of the optimal hyper-plane, non-linear optimization methods for SVMs, and decomposition methods for solving the convex optimization for SVM.

For instance, SVM is more appropriate for classifying non-linearly separable data [26], enables classification in multi-dimensional space [24], is capable of handling high dimensional data [27], generate high accuracy results [27], [24], and can be extend to regression problems. In addition, classification process of the SVM can be further improved by incorporating non-linear optimization methods to find the optimal solutions [27]. Comparison of SVM with the most commonly used neural networks is as follows:

- SVM are favorable for applications with limited availability of data samples compared to neural networks.
- The neural network has shown some significant performance over the past few years compared to SVM, but they are data hungry learning algorithms. In order to

train a neural network to achieve high accuracy output, we need massive amounts of data. However, for certain applications, the availability of data is limited.

- Also, training a neural network requires spatial property, which means the order of the input training samples determine the final output of the classifier. Changing the input order has little to no effect on SVMs.
- Tuning the neural network hyper-parameters for a specific application is time-consuming and a tedious process. On the other hand, we can analyze the hyper-parameters of SVM by examining the values and determine a certain threshold.
- Since our aim is to provide a hardware support, we are designing the embedded architecture on a hierarchical approach for both the classifier, which is beneficial to configure the design to utilize shared resources. Since, SVM and NN share some of the common operations; our hierarchical platforms can be easily modified to utilize the generic IP modules for either of the classifier.

Embedded hardware architecture and acceleration solutions are relevant to almost all the machine learning classifiers due to the recent accumulation of large-volume of data. However, most of the classifiers fail to process high-dimensional features in large-volume of data. Due these limitations, neural networks, support vector machines and random forest data classifiers are among the top three classifiers in the field of machine learning [32]. Although, these aforementioned classifiers are formulated to process high-dimensional data, the execution time increases exponentially with respect to the number of data samples. In order to improve the execution time, any optimization from software perspective would be insufficient. Customized hardware architecture is essential in order to reap actual benefits from these classifiers for real-time applications.

2.4 Existing research work on hardware support for SVM

We surveyed the existing research work on hardware support for the SVM classifier. In this regard, we investigated ways to utilize Field Programmable Gated Arrays (FPGAs) to design, develop, and implement high-dimension, large-scale data classifier. We performed an extensive investigation on the existing works on FPGA-based hardware architectures for CO-based SVM algorithms in the published literature. Additional detailed analysis of other existing works can be found in some survey papers such as [33] [34].

FPGA-based parallel processing hardware architecture was proposed for SVM using stochastic gradient descent (SGD) as the training method, in [35]. The authors demonstrated the scalability of the SGD approach for SVM in terms of fixed-point vs. single-precision floating-point computations. The hardware design was generated using the Xilinx System Generator design tool, and executed on Xilinx ML605 board with Virtex-6 FPGA. In this case, the synthesis results were obtained and reported, in terms of area, time, and throughput; however, the classification accuracy results were not reported. From the results, it is evident that parallelization led to the increase in occupied area, thus confirming that higher speedup due to parallelization, comes with the penalty of larger occupied area on chip. The proposed design could have demonstrated to execute datasets with more than 4 features/attributes, which is indeed a limitation when executing large volume of data with many attributes. Conversely, our proposed design can execute datasets with varying sizes and with any number of features/attributes.

In [4], an energy-efficient embedded binarized SVM architecture was proposed and implemented on an FPGA. The computation kernels were designed in C/C++ and transformed into HDL using Xilinx HLS (high-level synthesis) tools. The proposed

hardware design was executed on Xilinx Virtex-6/7 FPGAs. The results were obtained and reported, in terms of area, speedup, power, and classification accuracy. The FPGA's performance matric results (especially speedup and power) were compared with that of the CPU and GPU. From the results, it is evident that FPGA and GPU achieved significant speedup compared to the CPU. However, the power consumption of the GPU was significantly higher than that of the FPGA. These results illustrate that FPGA-based hardware architecture for SVM can achieve better performance-per-Watt, thus suitable for embedded devices with stringent power requirements.

An FPGA-based hardware accelerator was proposed for approximate SVM in [36], utilizing two approximation techniques, including precision scaling and loop perforation. The hardware was designed using Xilinx Vivado HLS tool, and executed on Xilinx Zynq7 ZC706 board. The results were obtained and reported, in terms of area, speedup, and classification accuracy. From the results, it is evident that the approximate computing led to higher speedup, but with the penalty of larger occupied area (or resource utilization) on chip, and lower classification accuracy. In some cases, the significant accuracy loss did not compensate with significant increase in speedup.

In [37], an FPGA-based hardware design was proposed for SVM classifier. In this case, three variable-size SVM models were implemented using different optimization techniques. The proposed hardware was designed using Xilinx Vivado HLS tool, and executed on Xilinx Zynq7 ZC702 board. The results were obtained and reported, in terms of area, speedup, power, and classification accuracy. Also, in this paper, the training phase was done offline on software; hence, the support vectors were pre-computed, and forwarded to the proposed hardware design, which is created only for the testing phase.

An FPGA-based parallel processing architecture was proposed in [38] for training phase of SVM using Sequential Minimal Optimization (SMO). The proposed hardware design was executed on Xilinx Virtex-6/7/Ultra-scale FPGAs. The synthesis results were obtained and reported, in terms of area, throughput, and speedup; however, the classification accuracy results were not reported. In this case, the authors utilized the hardware friendly kernel (HFK) for SVM training, which leads to reduction in precision of the floating-point operations. Although marginal loss in accuracy is acceptable for testing, utilizing HFKs for training would result in an inefficient construction of a hyper-plane during training.

In [39], a FPGA-based hardware-software co-design was proposed to accelerate the SVM algorithm by utilizing a two-level approach: first to optimize the global structure of the SVM; and second to refine it through the design exploration. The proposed architecture was designed using Xilinx Vivado HLS tool, and executed on Xilinx Zynq Zedboard. The results were obtained and reported, in terms of area, latency, and speedup; however, the classification accuracy results were not reported. As authors indicated, for high values of SVM parameters, the resource utilization (i.e., occupied area) increased significantly, which would be an issue for embedded devices with stringent area requirements. In this paper, the authors extensively discuss and analyze the advantages/disadvantages of utilizing the HLS tools to transform the designs written in C/C+ to HDL, thus providing insight into the HLS inefficiencies, which would be very useful when creating optimized hardware architectures in order to improve certain performance metrics, including the latency.

An FPGA-based coarse-grained reconfigurable hardware architecture was proposed in [40], for various machine learning (ML) algorithms, including SVM, decision trees, and artificial neural networks. The hardware was designed using Xilinx Vivado tool, and executed on Xilinx Virtex-7 FPGA. The results were obtained and reported, in terms of area, and speedup; however, the classification accuracy results were not reported. In this case, in order to change from one ML algorithm to another, authors claim that the reconfigurable processing nodes (RPNs) of the proposed architecture, can be reconfigured individually; however, no details are provided how this can be done. This requires partial reconfiguration of the FPGA; thus, adding significant complexity to the design process, which has not been addressed or discussed in the paper.

A scalable FPGA-based architecture was proposed in [41] to accelerate the SVM classification. The hardware was designed in VHDL, and executed on Altera Stratix III EP3SE260 board. The results were obtained and reported, in terms of speedup; however, the occupied area was not reported. Furthermore, in this paper, the authors only proposed the hardware design for the testing phase. Hence, the support vectors were pre-computed and stored in the on-chip memory for subsequent processing during the testing phase. The same authors proposed a design flow for the SVM training phase in [42].

An FPGA-based design was proposed for decision boundary conditions using multiplier-less kernel implementation technique for image processing in [43]. In this case, only the classification step of the SVM was implemented using the proposed multiplier less techniques to reduce power. However, the training process of SVM was computed offline using MATLAB, which in many cases requires the hardware support to improve the computational time. The support vectors obtained from the MATLAB was

stored in the FPGA's internal memory and utilized for decision boundary conditions. In these scenarios, incorporating the multiplier less kernel technique might be useful in order to reduce power but which in turn might add execution time for the overall decision conditions. The power consumption was reported in the range of 1.479 to 2.051W for the multiplier-less kernel implementation, which is high for decision boundary condition computations.

In [79], a hardware design based on the stochastic gradient descent algorithm for SVM was proposed. The authors demonstrate the scalability of the stochastic gradient descent approach of SVM for fixed-point vs. floating point computations. The hardware design was generated using the system generator tools. In regards to the system generator tools, details were not found related to any inefficiencies of the system generator tools as specified in [94]. Also, a brief comparison relative to the accuracy of the classifier would be applicable for exponent implementation using look-up table and other alternative approach. The comparison made in this paper could be in parallel in terms of seconds saved to improve the performance.

In [82], the paper investigates the potential performance gain for SVM kernels and decision boundary conditions. The authors measure the performance of these operations on CPU, GPU and FPGA. Based on the results analysis, the speedup gain of a GPU and FPGAs are significantly higher than a general purpose processor. However, the power consumption of the GPUs is 50 times higher than a FPGA. This shows that the FPGA-based designs can achieve a better perf/W.

An approximate computing approach to improve SVM performance was proposed in [83]. In this approach, algorithmic approximation such as precision scaling and loop

perforation was utilized. Approximate computing can result in a high speed performance and efficient power design at the cost of the accuracy of the machine learning model. Since machine learning models are expanding into many accuracy critical applications, it seems reasonable to consider a certain threshold for approximating. A speedup of 15x was illustrated with respect to the software counterpart. In order to analyze the details of the approximate computing, power consumption and computing time for generating support vectors may be beneficial.

In [84], a FPGA-based design was generated using the ultrafast high level synthesis (HLS) design methodology instead of classic hardware description language. The design achieves a speedup of 36.98. However, linear kernels and decision boundary conditions were implemented, but the support vectors were pre-computed. The details related to the HLS inefficiencies can be highlighted in order to generate a optimized design.

In [85], a parallel implementation of sequential minimal optimization for training was proposed. In this paper, the authors utilize a hardware friendly kernel to train SVM model. However, the hardware friendly kernels approach would be to improve the power consumptions but may lead to approximate computing. Specifically, for a decomposition approach, the accuracy depends on the numerical precision. Also, no details were found in regards to the exponent computations, memory management and decision boundary conditions.

In [94], a co-processor design was proposed based on two-level methodology exploring the usage of design space techniques to produce an efficient accelerator by addressing the HLS inefficiencies. Although, HW/SW co-design may not suitable for

embedded mobile applications, one of our objectives is to provide a dedicated and independent hardware modules to achieve a higher speed up. Using a HW/SW co-design, the paper illustrates a speedup gain of 78x, however, the power consumption would lead to lower performance/W. The paper provides useful insights on the HLS inefficiencies, which are useful to develop a more optimized hardware design to achieve a better latency gain.

In [97], a reconfigurable FPGA-based architecture was proposed for various machine learning algorithms such as Decision Trees, Support Vector Machines and Artificial Neural Networks. In this case, the paper aims to provide one universal solution for machine learning algorithm on hardware architecture. One universal solution seems a reasonable approach and by adapting the hardware for different applications, but the scalability to reconfigure might lead to increased place and route time with respect to the size of the application domain. Specifically, as the dimensions of the data to process in a machine learning increases, the choice of a specific machine learning solutions would be a tedious process along with the mapping the design on to the FPGA. However, the paper demonstrates a speedup gain of up to 66.71x, which illustrates the potential gain from a FPGA-based design, which can further be improved with optimized hardware architectures.

In [42], a FPGA-based architecture is proposed for training heterogeneous linearly separable datasets. Regarding the scalability issues suitable for higher-dimension and large-scale datasets are not addressed and based on the architecture provided, the computational elements accomplish the linear kernel task and does not extend its applicability to solve non-linear applications. It should be noted that the hardware support

utilizes PCIe for accessing the data; it might not be suitable for embedded applications to utilize a PCIe due to limited hardware footprints. Therefore, utilizing one of the high-speed communication protocols for the embedded platforms are efficient than large physical connectors to access data.

A GPU-based design was proposed for data classification using Gilbert's algorithms to evaluate the kernel functions [45]. The paper presented brief details about GPU and FPGA implementations, a brief internal architecture of the SVM training and testing process. Using CUDA, the GPUs are treated as a co-processor serving the host CPU. Based on the results, the speedup comparison of FPGA vs. GPU is linearly proportional to the dimensionality of the data. As the number of dimensions increases, the FPGA speedup increased linearly from 10 to 90x. Due to limited internal memory of FPGAs the speedup starts to saturate and decline for larger scale data sets. Since, GPU are high power consuming devices, power comparison for this approach would be highly applicable. Nevertheless, the FPGA showed better performance compared to GPUs. Also, in order to maximize the acceleration potential, GPUs need a host CPU to communicate, in turn adding the additional hardware resources. Therefore, for our design approach, we developed an efficient pre-fetching technique to overcome the issues mentioned above and provided independent and dedicated hardware design.

In [46], a comparison for the SVM implementation on FPGA and GPU was proposed; it did not detail the hardware implementation of the SVM. The paper provides a brief idea of hardware implementation for the data segmentation application using SVM. The paper provides results in terms of speedup and power consumption for FPGA vs. GPU design. Based on the result analysis, FPGA outperforms GPU by a factor of 3.5

times and consume 17x less power than a GPU. This significant power consumption of GPU is evident to choose power efficient embedded platforms for large-scale data applications. The results suggest that the power consumption for GPU based design is 250W and 15W for FPGA. Although, GPUs has high parallel processing capabilities but the cost and power constraints are a major threshold for large-scale data applications.

Another comparison study for GPU vs. FPGA was proposed in [47]. In this paper, a speedup comparison was performed on different hardware at a wide price range. The hardware 1 is the Tegra K1 GPU costing at around $200, hardware 2 is Zynq-7000 FPGA at $400 and hardware 3 is the Tesla P100 HPC server costs over $8000. Based on the different price points, FPGA performed ~10x better compared to the regular GPU whereas, the P100 HPC server outperformed FPGA by ~3x. Considering the cost of these devices, the speedup obtained is marginal. The paper did not detail the SVM optimizations or power consumptions regarding these hardware implementations.

In [48], a parallel implementation of FPGA as coprocessor was proposed for SVM. With this design, the speedup obtained was around 20x compared to CPU design consuming 10W power. Although, the design utilize PCI and FPGA as a co-processor to parallelize compute-intensive arithmetic operations, it is not suitable for mobile embedded platforms, which has stringent area and power restrictions. Considering these design constraints, our design is independent with less area and power suitable for mobile platforms. In additions, PCI module is suitable for static environment as compared to high speed AXI protocols. With the limited hardware footprints, our design is optimized and efficient embedded applications.

An FPGA-based implementation for SVM was introduced in [29] for telecommunication. In this case, only the SVM training phase was implemented on hardware. Some parts of the training phase, such as kernel and Q matrices, were also computed offline. In order for a data classifier to be efficiently utilized in real-world applications, it is imperative to design an adaptable training and testing modules on embedded platforms. Since the real-world is typically high-dimensional and large-scale data, training offline suitable for various static applications. However, it is not applicable for dynamic or mobile applications, in which the data access latency plays a major role.

In [30], a coprocessor was introduced only for the kernel matrix computation of the SVM training. However, it is imperative to implement all the stages of both the training and testing phases on hardware, especially for real-time machine learning applications. Accelerating the kernel matrix computations is a part of the SVM classifier, which may not be efficient for application such as small-scale linearly separable datasets. Therefore, implementation of a generic SVM module including both training and testing process seems pliable for real-world data applications.

In [31], a scalable FPGA-based architecture was proposed for the SVM algorithm. Although the proposed hardware classifier achieved a substantial speedup compared to its software counterpart, the design did not include techniques: to solve the constraint quadratic formulation, and to process and analyze the data in real-time.

An FPGA-based accelerator was proposed for a different SVM algorithm known as least square SVM [49]. In this case, the SVM training was processed online utilizing a run-time reconfiguration framework and parallel processing architecture. This improved the speedup but with the penalty in area, which is not feasible especially for small

footprint embedded devices. Also, details regarding the stream of data to be processed for real-time classification needs to be addressed with regards to memory and communication protocols.

From this investigation, it is evident that most of the existing works proposed hardware architectures either for testing or for training, but not for both. Furthermore, most of these proposed hardware architectures were not generic or parameterized. Also, most of these architectures were not designed with embedded devices in mind. None of these works proposed system-level architectures and associated techniques to facilitate real-time processing of machine learning applications. Consequently, the existing works did not report the corresponding system-level area, and did not consider the associated memory access latency while reporting timing/speedup. As a result, we could not make any direct performance comparisons with the existing works on FPGA-based hardware architectures in the published literature. In summary, from this investigation, and to the best of our knowledge, we could not find any similar work as ours, in the published literature, that provides FPGA-based hardware accelerators for CO-based SVM, especially on embedded devices, nor could we find any similar work that proposed system-level architectures, which is imperative for the machine learning applications in real-world scenarios.

Depending upon different approaches considered by many peers the following table below summarizes the existing research work over the past decade. Each article addresses specific problems to enhance the efficiency and adaptability of the machine learning algorithms for large-scale data applications. We have extracted the specific and

related details corresponding to hardware computing platforms to implement data classifier applications.

All the implementations addresses hardware implementation of the SVM classifier and to accelerate certain arithmetic computations, however designing a generic and independent classifier requires addressing the main constraints such as memory access latency, floating-point IPs, exponential implementation, dynamic algorithms suitable for real-world data sets irrespective of linearly or non-linearly separable datasets, kernels utilizations. Maintaining a specific DSP and logic resources ration might provide guidelines in regards to accelerate the computations within area constraints however, the real-time data processing must be taken into consideration for developing efficient embedded hardware architectures.

Unlike the existing hardware architectures for modules and sub-modules of SVM implementation discussed above, our FPGA-based hardware architectures are generic, parameterized, and scalable independent IP modules for both training and testing purposes. Our hardware designs are design for homogeneous and heterogeneous data sets, linearly and non-linearly separable data sets without changing the internal hardware architectures or affecting the occupied area of the chip. Based on the previous analysis on the existing architecture, our designs are optimized to consume significantly less power for the entire SVM process as opposed to partial implementation specified above. By providing generic and individual IPs for each stage, IPs can be used for any embedded applications that employ either entire SVM modules or any specific individual modules.

As mentioned in the literature review in section 2, the existing research work does not address major issues and challenges for designing a generic, independent data

classifier. Therefore, we identify the constraints related to embedded platforms and address them accordingly:

- Provide a system-level architecture for data classifier
- an efficient pre-fetching techniques to reduce memory access latencies
- implementation of Taylor series expansion to address IP modules limitations
- generic embedded hardware design for various types of datasets (homogeneous and heterogeneous, linearly separable and non-linearly separable)
- efficient decomposition implementation for convex optimization along with the boundary decision condition
- implement adaptable and independent training and testing modules for real-time applications
- accelerate both training and testing using parallel systolic array implementation
- Compare speedup performance with respect to CPU design, software counterparts as well as hardware design with/without systolic array instances.

Why Embedded hardware?

Embedded platforms have certain advantages compared to general-purpose processor such as:

- Low cost, less area and power requirements
- Faster real-time computations, better performance
- Customizable, reprogrammable.
- designed to perform a specific task independent or part of larger system
- High throughput to numerous data-intensive applications with critical time constraints

Why FPGA?

- Large number of LUTs, DSP blocks and a hierarchy of different memory sizes, providing high level of design flexibility
- Runtime re-configurability allows the design to be scalable and adaptive to different types of input data
- FPGAs provide numerous advantages such as high parallelization, better performance, efficient prototyping capabilities, lower power etc.

Table 1.01. Literature review

Ref	Contribution	Platforms	Resource utilization	Acc (%)	Speedup	Pow (W)
HW_ v1	Kernels, Decomposition, boundary decision, memory management	Xilinx Virtex-6 ML605	Slices-5216, (LUTs-12965, reg-12784), DSP-110, BRAM-118	100	CPU-3.1x, Software-74x	2.96
HW_ v2	Systolic Array	Xilinx Virtex-6 ML605	Slices-8390, DSP-236, BRAM-118	100	SW-107x	3.42
[50]	GPUs, efficient memory management	Intel i7,i9 GeForce RTX2070, GTX 1080Ti	N/A	100	GPU1 vs. GPU2: 140x	N/A
[35]	HW design generated using sys gen	Xilinx-Virtex-6	Slices-25790, DSP-450	100	CPU-319x,	N/A
[51]	Testing	Xilinx Virtex Ultrascale+ XCVU9P	N/A	N/A	N/A	N/A
[52]	Hyper-parameter, grid search	Intel i7-6700K & NVIDIA GTX1080Ti	N/A	100	N/A	N/A
[4]	Kernel & decision function	Intel i7, Nvidia GeForce GTX Titan X, Xilinx Virtex-6/7	LUT-12072, DSP-320	94	CPU-2220x, GPU-0.373x	3.2,5.4, 160[1]
[36]	Approximate computing, kernel	Xilinx Zynq-7 ZC706	N/A	96	SW -15x	N/A

[1] CPU, GPU and FPGA power consumption

	& testing					
[37]	HW/SW SoC, Linear kernel & Testing	Xilinx Zynq-7000 platform	3% resource util., Slices-13,830	97	SW - 36.98x	2.65
[38]	Training-SMO, shift-add, simulation	Xilinx Virtex-6 XC6VLX240T-1FF1156	Logic cells-106008 (70.3%)	N/A	SW-1312x	N/A
[53]	Incremental/Decrement SVM (IDSVM)	FPGA, MATLAB	N/A	N/A	SW-70%	N/A
[54]	Cascade SVM, kernels & testing	Xilinx Spartan-6 XC6SLX150T	LUTs-45758	84	SW-34fps	10.4
[55]	Pre-computation-kernel	Intel Xeon E5-2640v4, Nvidia Tesla P100	N/A	N/A	GPU vs. CPU-437x	N/A
[56]	Testing	HW/SW co-design Xilinx Zynq ZC702	2.7% resource utilization	97	SW-37x, GPU-7x	1.69
[57]	Testing, sys gen, Matlab-HW design	Xilinx Virtex-5	N/A	N/A	SW-60x Matlab simu.	5% less
[58]	Hyper-parameters using Bio-inspired opti.	Altera Cyclone IV EP4CE115F29C7	Logic Elements-44.7%, DSP-70	N/A	SW-3.67x	N/A
[59]	Testing	Xilinx Virtex-5	Sl. reg, LUT, DSP-11325,11467, 12	97	N/A	N/A
[60]	Two-level hierarchy for global & local matrix-matrix computation	Intel i7, NVIDIA GTX,1080, GTX980 Ti, GTX Titan black	GPU-61.65-96.98% of 2560,2816,2880 cores	100	GPU vs. GPU speedup-12x	N/A
[39]	Acceleration for efficient SVM HW co-proc.	Co-processor. Arm pro, Xilinx Zynq SoC	N/A	98	SW-78x.	N/A
[47]	Training-offline using LIBSVM-3.2	GPU, FPGA	N/A	97	GPU vs. FPGA-9.6x	N/A
[61]	Hybrid Zynq using HLS	HW/SW co-design, Xilinx Zynq	N/A	N/A	N/A	1.73
[62]	Multiple SVMs, partial out. combined &	Multiple proc.	N/A	100	CPU-35x	N/A

	filtered					
[40]	Reconfig. arch for DT, SVM, ANN	FPGA	N/A	N/A	SW-66.71x	N/A
[63]	IPM for based on Incomplete Cholesky Factorization	CPU-GPU cluster. Intel i7, NVIDIA GTX 480	N/A	98	CPU vs. GPU-36.68x	N/A
[64]	Training for cloud	Cloud	N/A	100	N/A	N/A
[65]	Efficient reconfig.	FPGA	N/A	N/A	Reconfig. -8x	N/A
[66]	Scaling to 12 cores	Xeon E5, Phi 7 110P, Tesla M2090,GK110	N/A	100	84x[2], 429x[3]	639[4]
[67]	Cascade SVM, hybrid arch, boosted by NN	Xilinx Spartan-6 XC6SLX150T	LUT-35532, DSP-59	80	40fps	9.9
[68]	kernel, hybrid working set in co-processor	Co-processor. Xilinx Virtex-7 VC707	N/A	N/A	CPU-25x	N/A
[69]	Multithread parallel framework for GPU	GPU. NVIDIA GeForce GTS 250	N/A	98	CPU vs. GPU - 15.21x	N/A
[43]	Training in Matlab, testing in HW	Co-processor. Matlab	N/A	N/A	N/A	2.05
[70]	Testing, Mac units, synthetic dataset	FPGA	N/A	99	N/A	N/A
[71]	Multiplier-less, kernel-systolic array	FPGA	N/A	N/A	N/A	N/A
[72]	Testing	XilinxV5 & Spartan-3E	1-3% slice registers	100	N/A	N/A
[73]	Convex quadratic prog. IPM	NVIDIA GTX-480	N/A	97	CPU vs. GPU-3x	N/A
[74]	IPM with ICF, Sherman Morrison Woodbury	Intel Xeon E5620 with 4xNVidia Tesla C2050	N/A	100	8x, 40x[5]	N/A
[46]	Testing, Gaussian kernel, co-design	GPU, FPGA	N/A	N/A	CPU-42x. GPU-(-	15, 250[6]

[2] Speedup of MIC and Ivy bridge
[3] Speedup in terms of GPUs and CPU
[4] (Gisette - 59W, Epsilon-104W, Dna-137W) + 502W system power
[5] GPU 8x the LIBSVM. P-PDIPM on GPU is 40x than S-PDIPM on CPU
[6] 15W for FPGA and 250W for GPU

					3.5x)	
[75]	Cascade SVM, hw reduction util.	Xilinx Virtex-5	43% less hardware resources	98	70fps and 5x	4.1
[76]	Online detection, testing, pre-computed SV	Xeon X5650, Xilinx Virtex-5 XC5VTX240	Occupied slices-32,679 + acc. cards	98	10Gbps	N/A
[77]	Posterior probability for testing, simulation	FPGA. Xilinx Virtex-5	N/A	97	N/A	N/A
[78]	Kernels	NVIDIA-8800GTS, NVIDIA GTS-250	N/A	N/A	CPU-9x	N/A
[79]	Simulation, training offline, SVs stored	FPGA-simulation. Xilinx Virtex-4 XC4VSX35	~200MHz, Slices (lin, non-lin):849-7261	98	SW-407x	N/A
[80]	SVs in memory banks, kernels, testing	FPGA. Xilinx Virtex-5 ML505	Slice LUTs: 57296	78	122fps	N/A
[42]	Training-linearly sep	FPGA	N/A	N/A	N/A	N/A
[81]	HW-SVR & SVM. Training-Matlab	Altera EP2C20 Cyclone II	Logic elements-75%	N/A	N/A	N/A
[41]	SVM – Testing	FPGA	N/A	N/A	CPU-3x GPU-7x	N/A
[82]	PC, frame grabber board + FPGA + GPU	Multi-platform	LUT-28616 for FPGA	95	N/A	N/A
[83]	Sparse matrix computations for || SVM	NVIDIA GeForce GTX470	N/A	N/A	GPU vs. CPU-133.8x	N/A
[84]	SMO, avoided numerical instability issues exists in traditional algorithms	Xilinx Virtex-4 XC4VLX100	~30% slices, DSP48-90%	N/A	Speedup relative ECU units-20x	N/A
[45]	Gilbert's Algorithm	GPU, FPGA	N/A	N/A	55x	N/A
[48]	Co-design	Co-processor	N/A	N/A	CPU-18-21x	10
[85]	Systolic Chain of PEs. Pre-computed SVs.	FPGA. Xilinx Virtex-5	N/A	88	~33fps	N/A

[86]	Two schemes- Logic elements & soft-core proc	Altera Cyclone II	Logic elements- 116750	100	Software- 41x	N/A
[44]	Training non-lin	FPGA	N/A	N/A	CPU-3x	N/A
[87]	Meta-heuristics, Variable neighborhood search	Embedded + FPGA + DSP + Intel Xeon 5320	N/A	N/A	N/A	N/A
[88]	Feed-forward phase	FPGA	N/A	5.9[7]	N/A	N/A

2.5 Chapter Summary

In this chapter, we discussed about the various means of hardware support for application specific operations. We provided framework details of machine learning techniques and applications. We elaborated on the mathematical procedure to construct a classification algorithm and optimization techniques. Existing work related to hardware support for the machine learning applications are presented. In the next chapter, we present the details about the architecture and implementation details for developing these complex algorithms on the embedded systems platforms.

[7] Error rate

CHAPTER 3

OPTIMIZED HARDWARE ARCHITECTURE FOR SVM CLASSIFIER

Our main objective for this research work is to provide efficient and optimized hardware architecture to accelerate the machine learning algorithm, specifically support vector machines. In the section, we discuss in detail about the design implementation starting with system-level architecture, pre-fetching modules to reduce the memory access time, custom IPs for classifications and performance evaluation.

3.1 Background: Convex Optimization and Support Vector Machines

In the following subsections, we discuss the mathematical representation to build an optimal hyper-plane, formulate the objective function to optimize the SVM algorithms and creating a feasible model suitable for large-scale datasets by incorporating the decomposition techniques.

3.1.1 Optimal Hyper-Plane

The SVM is a commonly used classification technique found in many different fields, such as digital channel equalization in signal processing, and protein structure prediction and cancer detection in medical diagnosis [26]. In this case, the concept is to formulate a hyper-plane with maximum margin width to distinguish between two classes [15], [31]. It is a supervised classification method [15], which often involves a training set of {xi, yi}, where x is the set of input data samples/vectors, i: total number of samples and y is the output label of the binary classifier, used for identifying the class of the data sample. These are represented in equation (1).

$$x = \{x_1, x_2, x_3 \dots x_i\} \forall i, x_i \in \Re^n$$

$$y_i \in \{-1, +1\}, \quad (1)$$

Each input vector x, constitutes different features of a dataset represented as, $x_i=\{x_{i1},x_{i2},x_{i3},\ldots,x_{in}\}$, where, n is the number of features. The formula for the hyper-plane [15] [24] [28] [25], and the decision function to determine the class are represented by the equations (2) and (3), respectively.

$$x_i \in \{x_{i1}, x_{i2}, \ldots, x_{in}\}, \text{ where, n is the number of features}$$

$$y_i(w.x + b) - 1 \geq 0 \text{ where, w: weight vector, b: bias value} \quad (2)$$

$$f(x) = \text{sgn}(w.x + b) \quad (3)$$

where, w is the weight vector, and b is the bias value.

The margin width of a hyper-plane can be obtained by projecting a unit normal vector ŵ to the optimal hyper-plane, as in equation (4) [15], [24].

$$\text{margin width} = \frac{2}{||w||} \quad (4)$$

The SVM can typically be extended to a multi-class classification [28], in which the output label is represented as,$y_i \in \{y_1, y_2, y_3, \ldots, y_m\}$, where, m is the total number of classes. As stated in [24], the multi-class training process can be performed using one-to-rest approach. With this approach, m different classes are trained independently, where one set of the input vectors forms the positive class, and the remaining input vectors form the negative class, in the hyper-plane. The classification process can further be carried out, similar to the binary classification.

3.1.2 Non-Linear Optimization

In order to improve the overall accuracy of the classification tasks, the margin width of the hyper-plane can be maximized using a non-linear optimization method [27], [28], [25]. With these methods, an objective function subjected to boundary constraints of

a hyper-plane can be derived from the equations (2) and (4) [15], [28]. The formula for the objective function is presented in equation (5), as follows:

$$y_i \in \{y_1, y_2, y_3, \ldots y_l\}, \text{ where, l: total number of classes}$$

$$\max_w(\text{margin}) = \max_w\left(\frac{2}{||w||}\right)$$

$$\text{Objective function:} \quad \min_w \frac{1}{2}(|w^2|)$$

$$\text{subject to} \quad y_i(w.x + b) - 1 \geq 0 \; \forall i \tag{5}$$

Employing the duality theory and the Lagrange multipliers [24], [23], [25], we can efficiently calculate the local maxima or the local minima of the objective function in equation (5) [27], [24]. From equation (5), the constrained optimization problem can be formulated similar to the non-linear programming/optimization [27], [25]. This constrained optimization problem can be presented as the primal and dual form as in equations (6) and (8) [24].

$$\text{Primal form, min } L(w, b) = \frac{1}{2}w^2 - \sum_{i=1}^{m} \alpha_i(y_i(w.x_i + b) - 1) \tag{6}$$

In order to obtain the minimum from equation (6), let's consider the point, where the gradient is zero [24]. The minimum value for the primal form can be obtained by applying the partial derivatives with respect to w and b to derive the following formulae in equation (7).

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i \tag{7}$$

Duality theory is useful for the constrained optimization problem, since it provides a convenient way to improve the data classification by utilizing the non-linear optimization approach [27], [25].

Dual form, max $W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j . K(x_i, x_j)$ (8)

The primal form (in equation (6)) can be solved using several methods [27], including Newton method, least squares algorithms, stochastic sub-gradient method, cutting plan algorithms, and interior point method. The dual form (in equation (8)) can also be solved using the decomposition methods and interior point method [27] [89] [90] [91]. The dual form has the advantage of utilizing mathematical kernels. Mathematical kernels are a set of algebraic transformation functions, which provides similarity information between data features [92], [93]. To use the mathematical kernels, the dual form depends on the pair of samples, such as K (xi. xj) as in equation (8). Utilizing the mathematical kernels, the SVM classification can be extended to non-linearly separable datasets [24] [94] [95] [93].

After obtaining the local minima from equations (6) and (8), the optimal value for α is evaluated to identify the support vectors [27], [24], [25]. The support vectors are the input vectors closest to the hyper-plane, and have an α value greater than zero ($\alpha > 0$) [27]. The dimensional co-ordinates of the support vectors determine the orientation of the hyper-plane. Any other vector may result in a closer to zero α value, which indicates that the data points have less impact on the orientation of the hyper-plane, and are also situated further away from the hyper-plane [24].

3.1.3 Convex Optimization

Typically for large-scale applications, due to high volume of data, the constrained optimization problem presented in equation (8) must converge to the minimum value, in order to find a best fit for constructing an optimal hyper-plane [27] [96] [97].

As stated in [24], for convex optimization, all local minima are considered as global minimum. In this case, due to the presence of noise in some datasets, the soft margin parameters such as l-norm error parameter (penalty parameter, C and slack variable (ξ)) must be considered for a better generalization of the primal form [24] [27]. Since the aforementioned non-linear optimization/programming consists of an equality constraint, the dual form (in equation (8)) can be reduced to the general form of convex optimization problem by integrating soft margin parameters [23]. Hence, the overall objective function in equation (8) can be modified to the following equation (9).

$$\text{Dual form, objective function: } \min_{\alpha} W(\alpha) = \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i P \alpha_j - \sum_{i=1}^{m} q^T \alpha_i$$

$$\text{subject to } 0 \leq \alpha_i \leq C$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \qquad (9)$$

The equation (9) is similar to the general form of convex optimization, as stated in [23], with the equality constraint and the box constraint, and can be written as equation (10) below.

$$\text{Dual form, objective function: } \min_{x} f(x) = \min_{x} \frac{1}{2} x^T P x - q^T x$$

$$\text{subject to } 0 \leq G x_i \leq h$$

$$A.x = b \qquad (10)$$

In convex optimization, the above objective function (in equations (9) and (10)) should converge in such a way that it satisfies the Hessian matrix condition [27] [25], which state that the contour of the convex plane should be continuously differentiable as in equation (11). Then, the optimal solution can be found where the gradient value is zero [24] [25].

$$\frac{\delta y}{\delta x} * \frac{\delta y}{\delta x} \geq 0 \quad (11)$$

In equation (10), the P matrix is symmetric and positive semi-definite; and both the objective function and the constraint function are convex [23] [25] [98]. Especially, for large-scale applications, the resultant matrices are dense, thus, difficult to solve. Therefore, the decomposition methods are often used to break down the convex optimization process by finding the two working sets [95] [99], as shown in equation (12). Proper selection of the "working sets" impacts the performance of the convex optimization algorithm and its convergence properties [27] [93] [25]. In this case, the process of finding the two working sets to determine the support vectors in the training set is a compute-intensive and an iterative process. Hence, by providing customized and optimized FPGA-based accelerators, we can dramatically enhance the speed-performance of these compute-intensive applications (or tasks).

The decomposition methods, such as sequential minimal optimization, are employed to solve the non-linear optimization applications/tasks by sequentially selecting the working set based on the proximal point with respect to the objective function (in equation (9)) [25]. Considering the various methods employed to solve the non-linear optimization, the Sequential Minimal Optimization (SMO) is the most popular, due to its ability to handle large-scale datasets efficiently and effectively [89] [100] [101]. In this case, at each iteration, the input vector x (in equation (9)) is divided into two working sets as $(x_k, \bar{x}_k)$, where, k is the current iteration count, x_k is the current input vector, and $\bar{x}_k$ is the previous input vector. Based on the specified starting point, the objective function (in equation (9)) is solved to converge to a minimum value [25]. More information, about the

equation (9), selection of the working sets, stopping criteria, and other methods to solve the non-linear optimization/programming, can be found in [23] [27].

For convex optimization (from equation (9)), a feasible point of (0, 0) is initially selected for the working sets, which is the first input data sample in equation (1). Then based on the direction of the gradient descent, the next working set is selected [23] [96]. Selecting the feasible point and the working set can impact the total time required to solve the convex quadratic optimization. As stated in [27], the solution for the convex optimization (in equation (9)) is found, by selecting the suitable working set, which satisfy the following criteria in equation (12).

$$\max_{i \in R(\alpha^*)} \left\{ -\frac{\nabla f(\alpha^*)}{y_i} \right\} \leq \min_{j \in S(\alpha^*)} \left\{ -\frac{\nabla f(\alpha^*)}{y_j} \right\} \qquad (12)$$

In equation (12), the R (α) and S (α) is the index sets used to characterize the descent direction, and allow to state the optimality conditions. Based on the descent direction, the objective function (in equation (9)) converges to the minimum α value, which is used to determine the orientation of the separating hyper-plane [27] [24]. In this case, constructing the hyper-plane establishes an explicit distinction between the different classes of the data.

3.2 Design Approach and Development Platform

In our designs, both hardware and software versions of various operations/tasks are implemented using hierarchical platform-based and modular-based design approaches to facilitate component reuse at different levels of abstractions. As illustrated in Figure 3.1, at the highest hierarchical level, our CO-based SVM algorithm comprises the training and testing tasks. During the training process, based on equation (9), the objective function is formulated to define the hyper-plane, to select a suitable

mathematical kernel, and to obtain the optimal solution. During the testing process, the decision function of the classifier is computed using the sign verification operation (in equation (3)). The aforementioned intermediate operations to process the training and testing tasks involve vector addition/multiplication, matrix computations, and various other arithmetic operations, which are placed at the lowest level of our platform-based design hierarchy.

During our early design phase, we investigate and utilize the integer units and also the double-precision floating-point units as our FPGA-based IPs. From these experiments, the integer-based designs exhibit results with quite low accuracy, whereas the double-precision floating-point designs occupy larger area on chip. These facts illustrate that the former might not be suitable for real-time applications, which typically require results with high accuracy; and the latter is not necessarily suitable for the embedded devices with the stringent area requirements. Furthermore, for the high dimensional input vectors, the optimal solution for α could not be reached (in equation (9)), utilizing the double-precision floating-point units, since these units not only create large on-chip logic resources requirements but also high latency requirements. Hence, we strive to approximate the results for the double-precision floating-point units, without compromising the accuracy of the results. As a result, we utilize the single-precision floating-point units, which create a tradeoff among the accuracy and area, and power. In this case, most of the lower-level operators are designed and implemented using the single-precision floating-point units in the Xilinx IP core library.

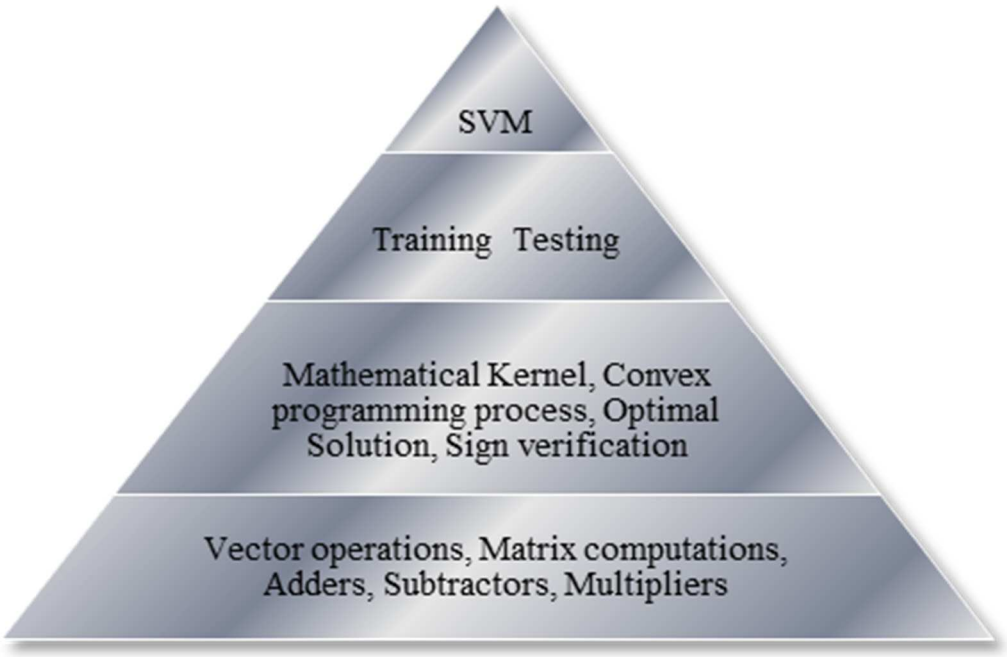


Figure 3.1: Hierarchical and modular-based design approach. Our design includes a hierarchy of abstraction levels, where higher-level operations utilize lower-level functional modules

3.2.1 Experimental Platform and Benchmark Datasets

All our hardware and software experiments are performed on the ML605 FPGA [102] [103] [104] development platform [105], which utilizes a Xilinx Virtex-6 XC6VLX240T-FFH1156 device. This development platform consists of large on-chip logic resources (37680 slices), 748 DSP48E1 slices, 512MB DDR3-SDRAM (Double-Data-Rate Synchronous Dynamic Random Access Memory), and 2MB on-chip BRAM (Block Random Access Memory) [106]. It should be noted that our hardware architectures for CO-based SVM were created in such a way to be generic, parameterized, and scalable; hence, without changing the internal architectures, our hardware designs can be executed on different embedded platforms, including the platforms with recent FPGAs such as Virtex-7 chips.

The vector computations are designed in such a way by integrating the available DSP48E1 slices to enhance the speed-performance. The large 512MB off-chip memory resources are useful to store large datasets, typically found in many real-time machine learning applications.

All our customized hardware modules are designed in mixed VHDL and Verilog [107], using Xilinx ISE 14.7 and XPS 14.7 design tools [108] [109] [110] [111]. They are executed on the aforementioned Virtex-6 FPGA running at 100MHz (in real-time) to verify their correctness and performance. The results and the functionalities of the hardware designs are further verified using the Modelsim SE, and Xilinx ISim tools [112] [113] [114] [115]. All our software modules are written in C++ and executed on the 32-bit RISC Micro-Blaze soft processor [116] running at 100MHz on the same FPGA. Xilinx XPS 14.7 and SDK 14.7 tools are used to design and verify the software modules [117] [118] [119]. Unlike the hard processors, the Micro-Blaze soft processor must be synthesized and mapped on to the configurable logic blocks of the FPGA. The performance-gain or the speedup is evaluated using the baseline software execution time over the improved hardware execution time. The hardware and software execution times are obtained from the AXI Timer [120].

The overall speedup is evaluated and reported using two different benchmark datasets obtained from the UCI machine learning repository [121]: Wisconsin breast cancer diagnostic dataset [21] and Ionosphere dataset for machine learning [122]. The benchmark datasets used to evaluate our proposed designs have two classes to account for binary classification task. The Cancer benchmark dataset comprises two classes, known as “Malignant” or “Benign”; and the Ionosphere benchmark dataset also consists

of two classes known as “Good” or “Bad”. The Cancer datasets consist of 569 samples (or vectors), each having 30 features (or attributes) to describe the characteristics of the cancer cell nuclei. These cancer cells are obtained using a fine needle aspirate and the main features are obtained from a digitized image. The Ionosphere dataset comprises a set of phased-array data obtained from 16 high-frequency radars. It has 351 samples, each having 34 features to characterize the complex electromagnetic signals. The total sizes of these two datasets are 68,280 bytes and 47,736 bytes, respectively.

3.3 Our Proposed System-Level Architecture

Figure 3.2 demonstrates the system-level architecture for our embedded hardware and software designs. Since 2MB on-chip BRAM, on Virtex-6 FPGA on ML605 board [123] [124] [125], is not sufficient to store the large amount of data commonly found in many machine learning applications, we integrate the 512MB DDR3-SDRAM external memory into the system. In this case, DDR3-SDRAM and the DDR3-SDRAM memory controller run at 200MHz, whereas the rest of the system is running at 100MHz. As illustrated in Figure 3.2, we utilize the AXI (Advanced Extensible Interface) bus [126] to facilitate the communication among the peripherals at the system-level.

During the initial software design phase, we configure the Micro-Blaze processor to have the maximum available cache memory of 128KB. However, this 128KB of cache memory is not sufficient to execute our software code and to process the data, since our code starts hanging. Hence, we vary the heap and stack size, and also increase the addressing of the cache memory to accommodate 256KB. This indeed resolves our cache memory constraint issue, although the Xilinx XPS tool still reports the size of the cache memory as 128KB.

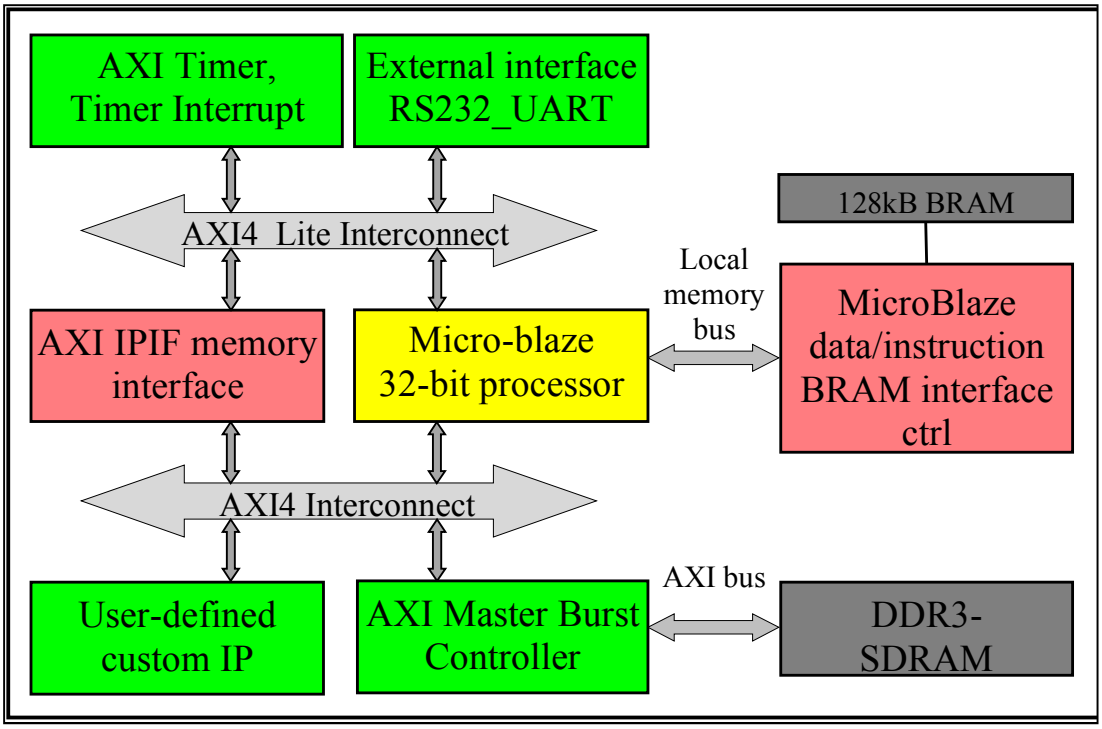


Figure 3.2: Our Proposed System-Level Architecture.

As shown in Figure 3.2, our user-defined/designed custom Intellectual Property (IP) communicates with the DDR3-SDRAM and the Micro-Blaze using the AXI bus through the AXI Intellectual Property Interface (IPIF) module, using a set of ports called the Intellectual Property Interconnects (IPIC) [126]. Typically, after the Micro-Blaze processor sends a start signal to our user-defined IP via the AXI4-lite bus and the AXI IPIF interface, our user-defined IP starts processing, directly reads/writes data/results from/to the DDR3-SDRAM via the AXI bus and the AXI Master Burst Controller, and sends a finish signal to the Micro-Blaze processor when the execution is completed.

For both our hardware and software designs, prior to computing the CO-based SVM algorithm, the original benchmark datasets are transferred from a host computer (i.e., desktop computer) to the ML605 development platform via the RS232 interface, and stored in the DDR3-SDRAM. In this work, the execution times (for both hardware and

software designs), reported in Section V, do not include this initial data transfer time via the RS232 interface. For our hardware design, we provide AXI4 burst/stream high-throughput interface (via the AXI bus and the AXI Master Burst Controller) for streaming the data from the DDR3-SDRAM to our user-defined IP for real-time processing. One of our design goals is to create our system-level architecture in such a way to train and classify a continuous stream of data using the AXI4 burst/stream interface. In a real-world application scenario, this feature enables proving a direct connection between our user-defined hardware IP and a camera, (for instance, in an autonomous car), in order to process the input data on-the-fly. This enables our hardware IP to perform the training and classification processes dynamically to cater to the ever-changing environment. By streaming the data and processing the data directly, reduces the amount of memory storage required for the embedded designs.

3.3.1 Our Proposed Pre-fetching Techniques and Top-level Architecture

Classification techniques, such as CO-based SVM, for machine learning applications, often involves processing large volume of data. This enormous amount of data must be stored in the external memory and transferred to the embedded platform for processing, since the on-chip BRAM is not sufficient to hold this large volume of data. This in turn leads to significant memory access latency, thus impacting the overall speedup of the hardware accelerators. From our previous work [127], [128], it was observed that a substantial amount of time was spent on accessing DDR3-SDRAM off-chip memory, which was a major performance bottleneck. Hence, it is imperative to address the memory access latency in our proposed embedded hardware architectures/accelerators.

In order to facilitate this endeavor, we create and integrate several design techniques to reduce the memory access latency as well as to enhance the speedup of our proposed embedded hardware architectures/accelerators. One of these techniques is the burst transfer. In this case, our user-defined hardware IP is designed to be enhanced with stream-in (or burst) data from the DDR3-SDRAM.

From Section III.b, as in Figure 2, the AXI4-lite and AXI4 interfaces act as glue logic for the whole system, including the MicroBlaze processor, internal peripherals, and the user-defined hardware IP. AXI4-lite is a single transaction memory-mapped bidirectional interface. In our embedded hardware accelerator, the MicroBlaze sends/receives certain control signals, and also monitors the status of the user-defined IP via the AXI4-lite bus and via the slave registers (or software accessible registers). These 32-bit slave registers are also used to send the SVM specifications, such as kernel type, dimensions of the input vectors, penalty/slack variables, to the user-defined IP, along with the initial memory address of the DDR3-SDRAM to access the datasets.

In addition to the AXI4-lite, the user-defined IP reads/writes data/results from/to the DDR3-SDRAM via the AXI4 bus and the AXI Master Burst Controller. AXI4 master burst is a high-performance memory mapped interface capable of transferring burst size of up to 256 data beats, which are compatible with 16, 32, 64, and 128 data width with a single address transaction phase. With this data width, we can transfer up to 1MB (2n-1 bytes) per cycle on IPIC command interface. By incorporating the AXI4 master burst capabilities (using the AXI Master Burst Controller), we dramatically reduce the memory access time for the SVM training and the SVM testing processes.

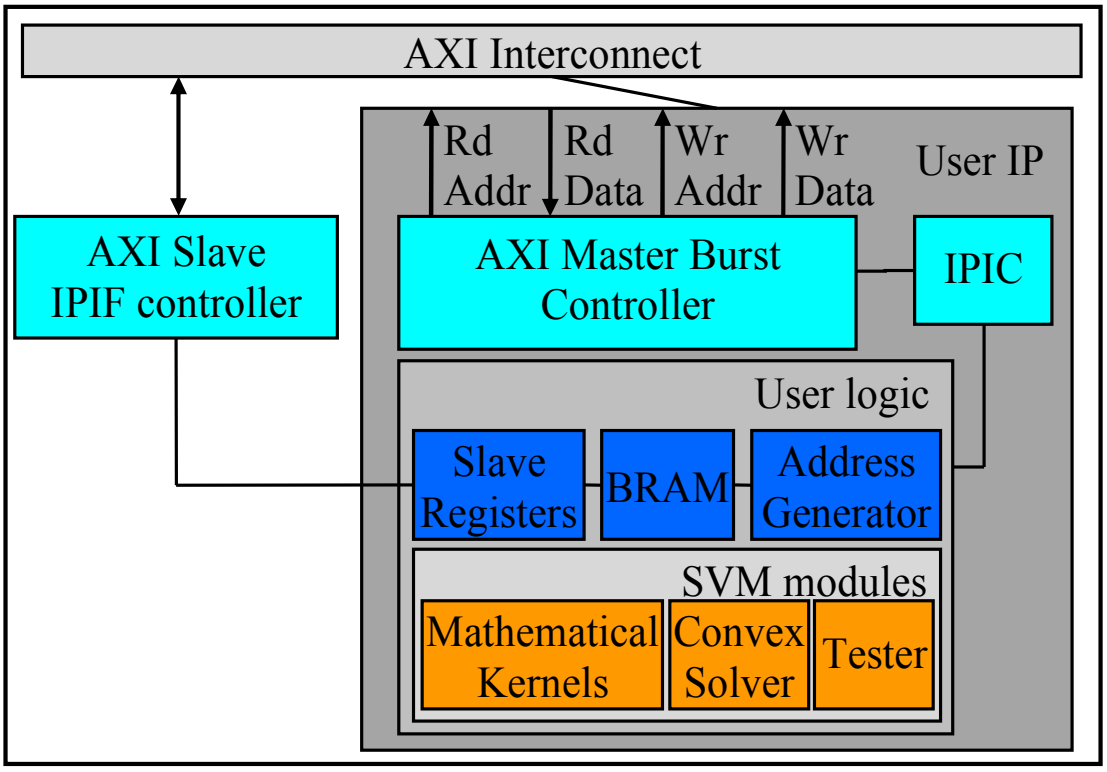


Figure 3.3: Pre-fetching Technique.

Apart from incorporating the AXI master burst, we create and integrate a novel and unique pre-fetching techniques to our used-defined hardware IP to further reduce the memory access latency. Our proposed pre-fetching technique and our proposed top-level architecture for the user-defined IP is illustrated in Figure 3. In this case, during the pre-fetching mode, our User Logic module (in Figure 3) determines the total number of bytes to be fetched utilizing the aforementioned SVM specifications provided by the MicroBlaze via the slave registers. Next, the address generator and the AXI Master Burst Controller configure the control signals of the IPIC with a suitable data width, a burst length, and a number of beats per cycle. Then, the AXI Master Burst Controller sends the aforementioned details about the data, as well as the "master read request" signal to the AXI interconnect core. Once the AXI master receives the "read request acknowledgment" signal from the AXI interconnect core, the user-defined IP can start

receiving the data from the DDR3-SDRAM, and store the data in the BRAM. Then the user logic can start the training process. In this case, the address generator is essential to index the correct values for matrix computations. After completing the training process, the AXI Master Burst Controller is designed to automatically setup for the write operation, in order to store the weight vectors and the bias values in the DDR3-SDRAM, for subsequent computations/analysis.

Our proposed pre-fetching technique is illustrated in Figure 3.3. During the pre-fetching mode, our user logic module (in Figure 3.3) determines the total number of bytes to be fetched utilizing the aforementioned SVM specifications provided by the Micro-Blaze via the slave registers. Next, the address generator and the AXI master burst controller configure the control signals of the IPIC with a suitable data width, a burst length, and a number of beats per cycle. Then, the AXI master burst controller sends the aforementioned details about the data, as well as the "master read request" signal to the AXI interconnect core. Once the AXI master receives the "read request acknowledgment" signal from the AXI interconnect core, the user IP can start receiving the data from the DDR3-SDRAM, can store the data in the BRAM [129]. Then the user logic can start the training process. In this case, the address generator is essential to index the correct values for matrix computations. After completing the training process, the AXI master burst controller is designed to automatically setup for the write operation, in order to store the weight vectors and the bias values in the DDR3-SDRAM, for subsequent computations/analysis.

Most of the existing techniques/algorithms for machine learning, including SVM, are typically designed in high-level programming languages such as python, and are

executed on general-purpose computers such as desktops and servers. These processor-based (software-only) algorithms, in their current form, cannot be executed directly on the embedded platforms/devices, since these devices have numerous constraints including stringent area and power, limited memory, increased speedup, and reduced cost and time-to-market requirements. Furthermore, today's machine learning techniques/algorithms are becoming more compute and data intensive, requiring more processing power. For instance, the processing time for the training would increase exponentially with the number of input data samples. Also, for smart and autonomous systems, the data processing and analysis must be done in real-time, in order to make split-second decisions.

Consequently, in order to satisfy the constraints associated with the embedded devices as well as the requirements of the machine learning applications, it is imperative to incorporate some applications-specific (or customized) hardware into embedded systems designs [130]. In this regard, Field Programmable Gate Array (FPGA)-based hardware is one of the most promising avenues to deliver machine learning applications on highly constrained embedded platforms [131], not only because FPGAs provides higher level of flexibility than ASICs (application-specific-integrated-circuits) and higher performance than software running on processor, but also due to its many attractive traits including post-fabrication reprogram ability, dynamic partial reconfiguration capabilities, and reduced time-to-market.

3.4 Embedded Architectures for Convex Optimization-Based SVM

In this section, we introduce novel, unique, and efficient embedded architectures (both hardware and software) for the convex optimization-based (CO-based) support vector machine (SVM) classification algorithm.

3.4.1 Embedded Software Design

Although our main focus of this chapter is to introduce embedded hardware architectures/accelerators for convex optimization (CO)-based SVM algorithm, we also create embedded software architectures for CO-based SVM mainly to evaluate our proposed embedded hardware designs. Our embedded software for CO-based SVM is designed and developed on MicroBlaze soft processor on the same development platform.

Prior to our embedded software designs, we design and develop the software for our CO-based SVM algorithm in C++ using the Microsoft Visual Studio development tools. This software design is executed on a desktop computer with Intel i7 processor running at 2.3GHz. Our results are compared and verified with the results from the open-source python code obtained from [132]. Both the C++ and python results are also used to verify our results from our embedded hardware and software architectures.

In order to cater to the resource constraint nature of the embedded devices, we significantly modify the aforementioned C++ software architecture, initially developed for the desktop computers. In this case, we create the codes leaner and simpler, in such a way to fit into the available program cache memory of the embedded microprocessor, i.e., MicroBlaze, without impacting the internal structure/flow and the functionalities of the overall CO-based SVM algorithm.

Algorithm:

Stage 1: Formulation stage

Input: Training set, m x n samples
m x 1 output labels
Specify C, ξ, γ, degree, kernel

Output: for i, j = 1, 2 ... m {
Kernel matrix computation
$K(x_i, x_j) = \Phi(x_i).\Phi(x_j)$ **(a)**
Compute P, q matrix from eq. 8
P = K [y[i] * y[j]]
q = [1, 1 ... 1].T **(b)**
subjected to constraints A, b, G, h **(c)**
$A = [y]_{1*m}$, $b_{const} = 0$
$G = [1,..]_i$, h = [C]}

Stage 2: Convex optimization and decomposition

Input: Set 2 working set, initial a feasible point (0, 0)
$\alpha = 0$, Grad, $Grad_{set} = 0$
Maximum iterations = 1000
Threshold = 0.0001

Output: For iterations from 1 to max iterations
for i, j = 1, 2, ... , m {
for k = 1, 2 {
Based on working set and gradient descent direction
$offset = -\frac{Grad}{P_{set\,1}+P_{set\,2}+2y_iP_{set\,1}}$ **(d)**
$\alpha_{new} = \alpha_{prev} + offset$ **(e)**
$dev = \alpha_{new} - \alpha_{prev}$ **(f)**
$Grad = dev*_{Pset1,set2}$ **(g)**
Reconstruct gradient
If(Grad≥0) Pset1[t:m] **(h)**
else Pset1[0:t]
Parameters update $\alpha_{set1,set2}$, Grad, $Grad_{set}$
}
Reiterate until maximum iterations
}
If α > Threshold
Assign support vector label, sv **(i)**
else α = 0
$bias\ value = \frac{\sum y_{sv} - \sum \alpha y_{sv} K(x_i, x_j)}{n_{sv}}$

Stage 3: Testing process

Input: Input test samples

Output: for i, j = 1, 2 ... m { **(j)**
Determine the class w.r.t separating hyper-plane
$Y_{predicted} = \alpha y_{sv} K(x_i, x_j) + b$
$Y_{predicted} > 0\}$
Based on sign of the decision function
classify to the corresponding class.

Figure 3.4: Software and Functional Flow for CO-Based SVM Algorithm

During our embedded software design phase, we encounter several issues due to stringent constraints of the embedded devices. One of the major issues is due to the limited memory resources. In this case, certain functionalities of the normal C++ programs, executed on desktop computers, can not be directly designed and implemented on the embedded devices. For instance, importing a pre-processor directive for vector computations, from the desktop computer to the embedded devices, resulted in memory limitations issues. In this case, we design a compact function in software that is capable of performing the vector computation efficiently and effectively.

Furthermore, for our embedded software designs, the MicroBlaze processor is configured to have the maximum available cache memory of 128KB, from which 64KB is used for the Instruction Cache and 64KB is used for the Data Cache. However, this 128KB of cache memory is not sufficient to execute our software code and to process the data, since our code starts hanging. To resolve this issue, initially, we vary the heap and stack sizes; and also increase the addressing of the cache memory to accommodate 256KB. This indeed resolves our cache memory constraint issues, although the Xilinx XPS tool still reports the size of the cache memory as 128KB.

Our embedded software architecture for the convex optimization based SVM algorithm comprises three stages. These three stages as well as the functional flow of our embedded software design are presented in Figure 3.4.

3.4.2 Embedded Hardware Architecture for CO-based SVM Algorithm

In this sub-section, we introduce our novel, customized, and optimized embedded hardware architecture for the convex optimization based SVM algorithm. In this case, we examine and analyze the functional flow of the aforementioned algorithm. Subsequently,

we partition this complex algorithm into three stages (SVM Module in Figure 3.3) to simplify the design process. The operations of these three consecutive stages are: mathematical kernels, convex optimization (or convex solver), and testing. It should be noted that the solver is considered as the hardware IP to perform the optimization stage.

In this research work, we create our customized and optimized embedded hardware architectures for each stage as separate modules. The hardware designs for each stage comprise a data-path and a control path. The control path consists of finite state machines (FSMs), and manages the control signals of the data-path and the BRAMs. We also design a top-level module (i.e., SVM Module in Figure 3.3) to integrate the three modules for the three stages. The top-level module provides necessary communication/control among the three stages. The control path of the top-level module also consists of several FSMs, multiplexers, and tri-state buffers to control the timing, routing, and internal structures/functionalities of the designs.

The three stages of the CO-based SVM algorithm are executed non-sequentially to utilize the parallel processing nature of the FPGA-based hardware. Initially, the first stage, i.e., the mathematical kernel, is processed until a certain amount of results is obtained from this stage. Then the p and q matrices (in equation 9), in the second stage, i.e., the optimization (or the solver), is computed on the aforementioned kernel results, while the remainder of the mathematical kernel is being proceed. In this case, the execution time to compute the p and q matrices is typically less than the execution time to compute the mathematical kernel; and the former depends on the results of the latter. Hence, we design and develop a simple counter to create a time delay in order to wait until the kernel has processed at least 50 training data samples, i.e., closer to 10% of the

total data size, before starting the second stage. Once the 10% of the mathematical kernel results (from stage 1) are available, stage 2 computation starts. At this point, both the stages 1 and 2 are executed in parallel. For all the stages, the intermediate/final results are stored in the BRAM, and after all three stages are processed, the final results are written to the DDR3-SDRAM.

The internal architectures of these three stages of the convex optimization based (CO-based) SVM algorithm are detailed in the following sub-sections. These internal architectures are customized and optimized in such away by exploiting the inherent parallelism and pipeline nature of the CO-based SVM algorithm.

3.4.2.1 Stage 1: Mathematical Kernels

In the first stage of our embedded hardware design, we select and perform a suitable mathematical kernel. As stated in [94], linearly inseparable vectors in the input space can be transformed to linearly separable vectors in the feature space by mapping the data points to a higher dimensional space. This transformation can be performed with mathematical kernels (including linear, polynomial and Gaussian kernels), since the bound does not depend on the dimensionality of the space for SVM [24]. This is an efficient way to obtain a well-defined separating hyper-plane [92]. Mathematical kernels are a set of algebraic transformation functions, which provides similarity information between data features [93]. In order to utilize the mathematical kernels, the mapping function (Φ(x)) in equation (16) must satisfy the Mercer's condition [92], which states that the inner product of the two input vectors must be defined for all the features, as represented in equation (13) below.

$$K(x_i, x_j) = \Phi(x_i).\Phi(x_j)$$

$$\int \int g(x_i)K(x_i, x_j)g(x_j)\, dx_i\, dx_j \geq 0 \quad (13)$$

In this research work, we decide to create customized and optimized embedded architectures for the Linear, Polynomial, and Gaussian Radial Basis Function (RBF), since these are the three most popular mathematical kernels used for the SVM algorithm. The equations for these three mathematical kernels are as follows [15]:

Linear Kernel: $K(x_i, x_j) = x_i . x_j$ (14)

Polynomial Kernel: $K(x_i, x_j) = (c + x_i . x_j)^d$ (15)

Gaussian RBF Kernel: $K(x_i, x_j) = e^{-\gamma(x_i - x_j)^2}$ (16)

The datapath for the polynomial kernel is illustrated in Figure 3.5 (corresponding to modules 2), whereas the datapath for the linear kernel is the dotted lines of Figure 3.5 (corresponding to modules 1). As shown, the datapath of the linear kernel consists of a multiplier, adder, and an accumulator register with the feedback loop to the adder, whereas the datapath for the polynomial kernel has a second adder and a power module.

The result of the linear kernel is the dot product operation of the two input data samples. In this case, initially, the first two elements of the two data samples are read from the BRAM, which is the first two elements of the first row, and the multiplication operation is performed, followed by the accumulation operation on each multiplier result. This process will continue until this multiplication and accumulation (MAC) operation is performed on the last two elements of the two data samples. Then the final result of the MAC operation is forwarded and stored on the BRAM for subsequent analysis/computations. As depicted in Figure 3.5, the modules in the dotted line (in the green box) comprise the MAC operation.

For the datapath for the polynomial kernel, the inputs to the second adder (Add_2) are the final result from the linear kernel as well as the coefficient c. This addition result goes through the power module to perform the "power of d" on the addition results. In this case, we create a hardware module for the power function using a simple loop to iterate the multiplication based on the specified degree (d) value. Although this degree value is parameterized in our design, a commonly used quadratic kernel is performed by considering this degree value as two (2). The final result of this power module (Pow in figure 3.5) is also forwarded and stored on the BRAM for subsequent computations.

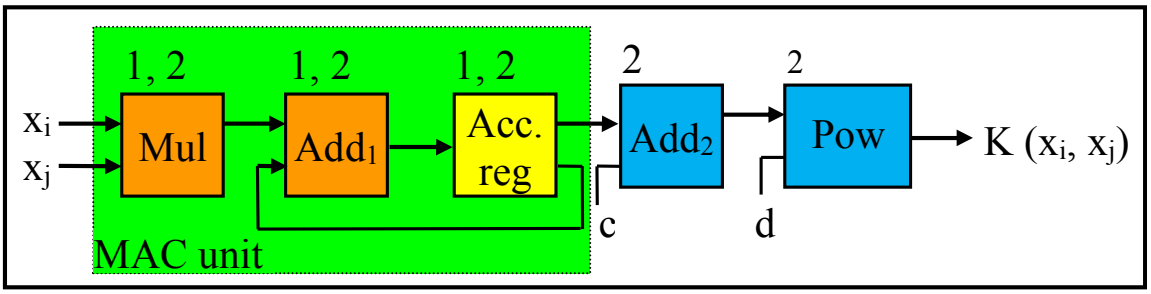


Figure 3.5: Datapath for Liner and Polynomial Kernels

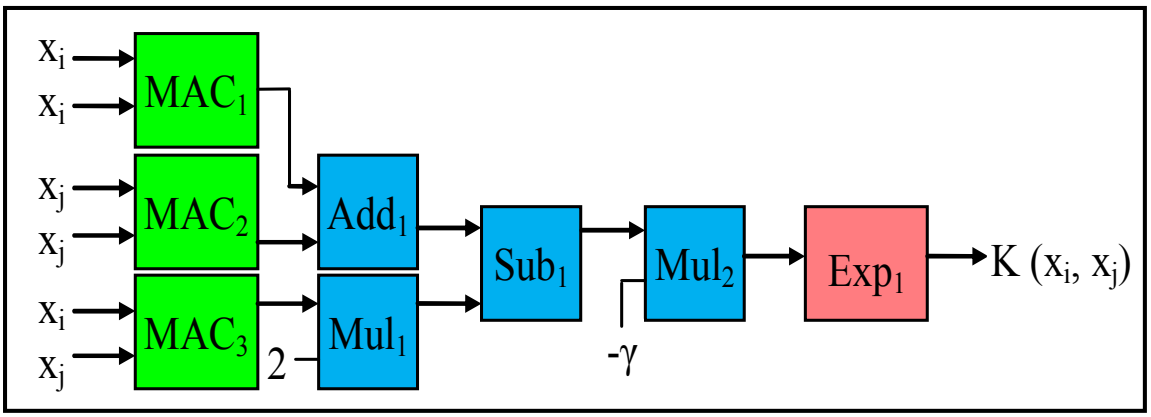


Figure 3.6: Datapath for Gaussian Radial Basis Function (RBF) Kernel

The datapath for the Gaussian kernel is demonstrated in Figure 3.6. The Gaussian radial basis function (RBF) kernel is the most popular among the aforementioned three mathematical kernels. The datapath comprises 3 MAC modules to perform part of the

equation (16), which is $(x_i\text{-}x_j)^2$, expanded into xi2+xj2-2xi.xj, which requires three dot product operations. Next, the addition operation is performed on the results of the two square operations, while the result of the MAC3 operation is multiplied by two. Then the multiplier result is subtracted from the result of the addition operation. The subtracted result is multiplied by the parameter known as gamma (γ). As stated in [132], this gamma (γ) parameter determines the influence of a data sample on the separating hyper-plane. In this case, the gamma (γ) value typically varies from 10-3 to 10+3 as needed. In this research work, as detailed in Section 3.4, we vary the gamma (γ) value from 2x10-3 to 2x10+2 for both our embedded hardware and software designs. As shown in Figure 3.6, the result of the multiplier operation (with $Mult_2$) goes through the exponent module to obtain the final result of the Gaussian RBF kernel. In this case, we create a parameterized exponent hardware module based on the Taylor series expansion, which is represented by the equation (17) [133] [134]. The final result of this exponent module is forwarded and stored on the BRAM for subsequent computations.

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} \ldots \quad (17)$$

The output size of the matrix for the kernel computation depends on the number of input data samples (m). Thus, the size of the K (x_i, x_j) matrix is m x m. After processing 10% of the data for the kernel computation, the convex optimization process is initiated. The computation to track 10% of the data processing is implemented using a simple counter. The kernel computation is necessary to perform all iterations of the next stage, which is the convex optimization. Hence, initially, the results of the mathematical kernel are stored on the BRAM to ease the iterative process of the convex optimization.

3.4.2.2 Stage 2: Convex Optimization

The optimization stage is the most complex operation among the three stages of the convex optimization based (CO-based) SVM algorithm. In order to reduce the complexity, we divide stage 2 into three phases: parameter initialization, convex optimization, and bias value computation. In this stage, the dual form of the SVM (in equation (8)) is utilized to formulate the general convex optimization as shown in equations (9) and (10).

3.4.2.2.1 Parameter Initialization Phase

During the parameter initialization phase of stage 2, several parameters in equation (9) are computed including the objective function parameters (i.e., P, q), constraint parameters (i.e., G, h, A, b_{const}), and other parameters (α, Grad, feasible point). In the convex optimization phase, α value and the Grad value are evaluated all the iterations using the sequential minimal optimization (SMO) decomposition method (detailed in chapter 2). Once the maximum number of iterations is reached, the bias value b is computed in the final bias value computation phase. This bias value is used to determine the intercept of the hyper-plane.

In this chapter, for stage 2, we design a generic convex optimization solver utilizing the same naming convention for the parameters as the general form of convex optimization from equation (10). In this case, the naming conventions b and b_{const} are the bias value (in equation (2)) and the constraints value (in equation (10)) respectively. The aforementioned objective function and constraint parameters are computed in Stage 2 (phase I), as shown in Figure 3.4 (steps (b) and (c) respectively).

Objective function parameters: The P parameter (in equations (9) and (10)) is an m x m matrix, which is computed using the dot product of the output labels y_i. y_j. The y variable is the output label obtained from the dataset, and y represents the class of the data sample, as in equation (1). In our design, during stage 1, y is typically pre-fetched to the BRAM with the input samples. During stage 2, the result of the dot product (y_i. y_j) is multiplied with the result of the kernel matrix K (xi, xj), in order to obtain the P matrix (in equation (18)). In this case, as illustrated in Figure 3.7, the modules to compute the P matrix consist of a MAC unit and a multiplier (i.e., MAC_1 Unit and Mul_1 respectively, in figure 3.7). The elements of the P matrix are stored in the BRAM. The elements of the P matrix in the BRAM are accessed using two separate address generators, in order to create two sets of matrices Pset1 and Pset2. As stated in [15], for big data analysis, the size of the P matrix increases in squared term with the increasing number of input data samples; thus, performing the convex optimization using conventional methods such as interior point methods become computationally challenging. To overcome this issue, a specific decomposition method in [95] is employed to perform the convex optimization. With this decomposition method, at any instance of time during the optimization process, two working sets are selected. This decomposition method is detailed in the subsequent convex optimization phase. The q parameter (in equations (9) and (10)) is an mX1 matrix. The q matrix is an array of ones, as in equation (19). For simplicity, during the design phase, each element of the q matrix is kept as constants of 1s.

$$P_{m,} = K\begin{pmatrix} y_0y_0 & y_0y_1 & \cdots & y_0y_m \\ y_1y_0 & y_1y_1 & \cdots & y_1y_m \\ \vdots & \vdots & \ddots & \vdots \\ y_my_0 & y_my_1 & \cdots & y_my_m \end{pmatrix} \tag{18}$$

$$q_{m,1} = (1, 1, \ldots, 1).T \tag{19}$$

Constraint parameters: In equations (9) and (10), the objective function is subjected to the box constraint (which comprises G and h parameters), and the equality constraint (which consists of A and b const). In this case, α parameter is a 1Xm matrix in equation (10), which is same as the output label y in equation (9). As shown in equation (10), b const parameter in equation (9) is set to zero. Also, bconst parameter in equation (10) is set to zero in equation (9). In addition, G parameter in equation (10) is a constant value of 1. The maximum threshold of the box constraint is h parameter in equation (10) (i.e., C parameter in equation (9)). C is considered as the penalty parameter, which is a user-defined vector provided during the parameter specifications. The value of C impacts the overall speed-performance of the CO-based SVM algorithm. This impact is illustrated in Section 3.4.

Other parameters: α parameter in equation (9), which corresponds to x in equation (10), is crucial to identify the support vectors in the CO-based SVM algorithm. Since the coordinate dimensions of the support vectors determine the orientation of the hyper-plane, α values for all the input samples are initialized to zeros, prior to performing the convex optimization. The goal of performing the convex optimization is to compute the minimum value for α for each input sample. Based on the aforementioned threshold value (C), the input samples with α value greater than threshold value, will be considered as the support vectors. Furthermore, as discussed above, since the size of the P matrix increases exponentially with the increasing number of input samples, two working sets (or input samples) are selected and utilized to compute α in all the iterations. In this case, for the initial starting point, a feasible set of (0, 0) is selected, which is the first input samples of the dataset; and the Grad (gradient) value is initialized to zero. The slope of

the gradient to the minimum value typically corresponds to the gradient descent direction; hence, the gradient value is utilized to select the next working set.

3.4.2.2.2 Convex Optimization Phase

After the parameter initialization phase, the convex optimization phase is performed. During the convex optimization phase, five operations illustrated in steps (d) to (h) (in Figure 3.4) are performed. The datapath for the convex optimization is shown in Figure 3.7, which consists of several adders, multipliers, subtractions, MAC modules, dividers, accumulator registers, comparators, and multiplexers. The convex optimization process (as in equation 9) involves finding the minimum α value. In this case, α value is computed using the Add_1 to Add_4, Mul_1, Div_1, and M_0 (multiplexer) modules, as illustrated in fig 3.7. In order to find the minimum α value, additional comparators, multiplexers, and reconstruct gradient modules are utilized.

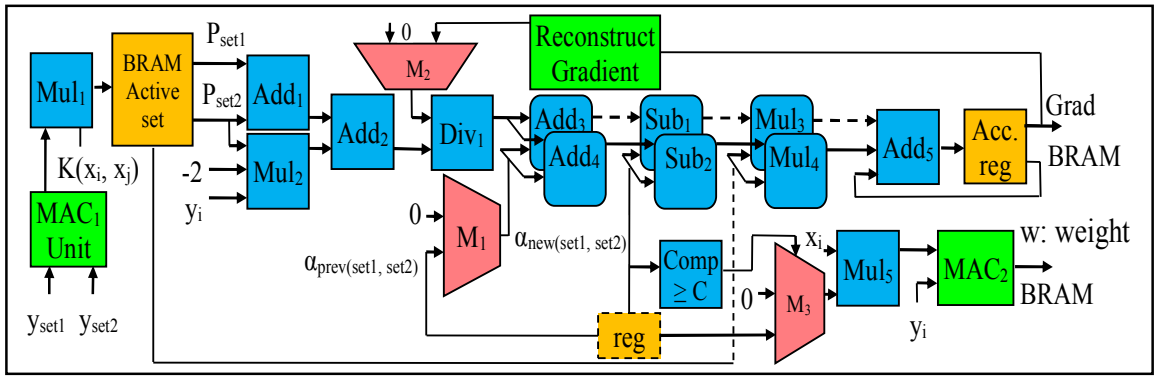


Figure 3.7: Datapath for Convex Optimization

Initially, during the first iteration to find the minimum α value, the two input samples, P_{set1} and P_{set2} (stored in the BRAM) are accessed using the two separate address generators, simultaneously. Next, the result of the addition operation (with Add_1 in Figure 3.7) of P_{set1} and P_{set2} are added (using Add_2) to the result of the multiplier (i.e., Mul_1), which corresponds to the step (f) in Figure 3.4. Then, the gradient value is divided

(with Div_1) by the result of the second addition operation (with Add_2). Since the Grad (gradient) value is initialized to zero, the initial result of the division is also zero. In our design, the computation of the gradient parameter is modified using the reconstruct gradient module in Figure 3.7. The internal architecture of the reconstruct gradient module is demonstrated in Figure 3.8. During the first iteration, the value of α is zero. The output (or result) of the division operation (with Div_1 in Figure 3.7) is considered as the offset value, which in turn is used to update α value. Apart from the first iteration, where the initial α value is zero, from the second iteration onwards, the aforementioned offset value, is added to α value, computed in the former iteration. For instance, the offset value produced from the second iteration is added to α value generated from the first iteration, and so on. In this case, two addition operations are performed in parallel (using Add_3 and Add_4), in order to generate new α values for each working set. The new α values are stored in the temporary register (i.e., reg in Figure 3.7), to be used for the future iterations. Next, two subtraction operations are performed in parallel (using Sub_1 and Sub_2), to find the difference between the new α value and former α value, which provides the deviation between the successive iterations. Then, the new α values are multiplied in parallel (using Mul_2 and Mul_3), with P_{set1} and P_{set2}, to find the gradient value (Grad). These Grad values are stored in the on-chip BRAM for subsequent iterations and operations. The aforementioned operations are illustrated in steps (d), (e), (f), and (g) in Figure 3.4.

Finding the minimum value, for the objective function in equation (9), is an iterative process. This process continues until it reaches the maximum number of iterations, typically defined by the user. Once the maximum iteration is reached, the α

values are compared with the user-defined threshold value (C) of 10^{-3} using the Comp module. In this case, the α values that are greater than the threshold value are considered as the support vectors, and the α values that are less than threshold values are discarded. These support vectors are important to determine the orientation of the hyper-plane. After obtaining the optimal solution for α (i.e., the minimum α value from equation (9)), this optimal α value is forwarded via the multiplexer (M_3) to a multiplier (Mul_5) followed by a MAC module (MAC_2) to compute the weight vectors (w) in equation (7). These weight vectors and the α values are stored in the BRAM as well as in the DDR3-SDRAM for subsequent testing stage.

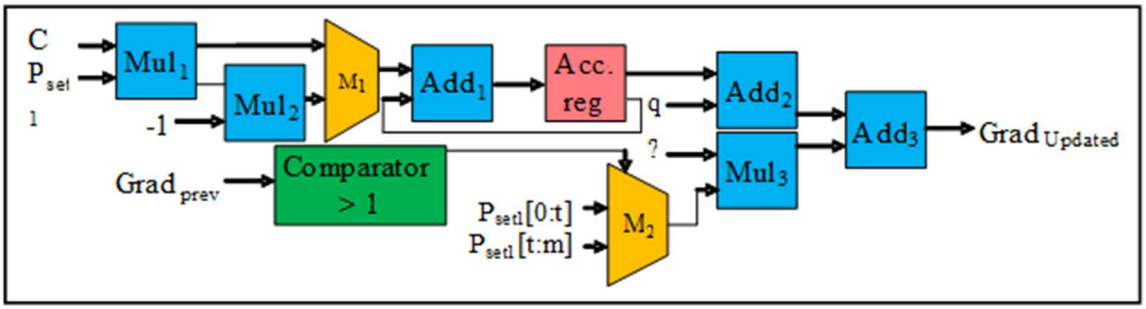


Figure 3.8: Internal Architecture of Reconstruct Gradient Computation

The internal architecture (or the datapath) of the reconstruct gradient computation is demonstrated in Figure 3.8. In case, if the optimization does not converge to a minimum α value during the iterative process of convex optimization (i.e., steps (d) to (h) in Stage 2, Figure 3.4), we utilize the reconstruct gradient module (in Figure 3.7) to adjust (or update) the value of the gradient parameter (Grad), and repeat the convex optimization process. Adjusting the values of the gradient parameter depends on the α values. As illustrated in Figure 3.8, the comparator module checks whether the gradient descent direction, of the current working set, is positive or negative. In this case, for the Pset1, if the gradient descent direction is positive, then the remaining Pset1 values (from t

to m) are assigned from the current number of active sets to the remaining number of data samples; and if the gradient descent direction is negative, then the working set is reset to select from the initial set (from 0 to t) as shown in step (h) in Figure 3.4. Next, the resulting Pset1 is selected via a multiplexer (M_2), which is multiplied (with $Mu1_3$) with the α value. Then the addition operation is performed (using Add3) on the results from Add2 and the results from Mu13, in order to obtain the restructured Grad value. This updated Grad value is stored in the BRAM for subsequent iterations and analysis, i.e., the "Grad" signal as shown in Figure 3.7.

3.4.2.2.3 Bias Value Computation Phase

The last phase of Stage 2 is the bias value computation phase. The bias value b in equation (2) is also known as the offset value [15], represents the intercept of the hyper-plane with respect to the origin. The bias value is computed with the following equation (20) [27]:

$$\mathrm{b} = -\frac{1}{2}\left[\max_{\{\mathrm{i}|\mathrm{y_j}=-1\}}\left(\sum_{\mathrm{j}=1}^{\mathrm{m}} \alpha_\mathrm{i}\mathrm{y_i}\mathrm{K}(\mathrm{x_i},\mathrm{x_j})\right) + \min_{\{\mathrm{i}|\mathrm{y_j}=+1\}}\left(\sum_{\mathrm{j}=1}^{\mathrm{m}} \alpha_\mathrm{i}\mathrm{y_i}\mathrm{K}(\mathrm{x_i},\mathrm{x_j})\right)\right] \quad (20)$$

$$b = \frac{\left(\sum_{i=1}^{m} y_i - \sum_{i=1}^{m} \alpha_i y_{sv} K(x_i, x_j)\right)}{number\ of\ support\ vectors, n_{sv}} \quad (21)$$

Equation (20) is modified to include the sum of all the support vectors and then divided by the total number of support vectors to obtain the mean value as in equation (21) [132]. As illustrated, equation (21) provides the average values for all the support vectors, whereas equation (20) identifies the max and min value for each class support vectors. The internal architecture (or datapath) of the bias value computation is demonstrated in Figure 3.9, which comprises a MAC module, adder, multiplier, subtractor, divider, and an accumulator register with a feedback loop to the adder. As

shown in Figure 3.9, the results (elements) of the kernel matrix (K (xi, xj) obtained from Stage 1) is multiplied with the α value (obtained from Stage 2). The result of the Mul module is multiplied with the y_{sv} value (which is the output label in equation (1) corresponding to the support vector) and summed using the MAC module. The datapath of Mul and MAC corresponds to the second summation term in the equation (21). The output label of the support vector (y_{sv}) is passed through the Add module and accumulator register (as in Figure 3.9), to obtain the first summation term in equation (21). Finally, the result of the MAC is subtracted from the result of the accumulator register, in order to obtain the numerator in equation (21). Finally, the result of the subtractor is divided by the total number of support vectors (n_{sv}) to obtain the bias value. The final bias vale b is stored in the on-chip BRAM as well as in the DDR3-SDRAM for subsequent analysis.

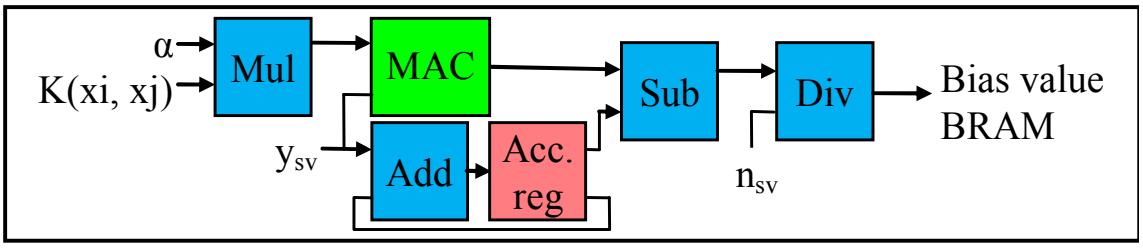


Figure 3.9: Internal Architecture of Bias Value Computation

3.4.2.3 Stage 3: Testing

Our stage 3, which is our final stage, is the testing process. Typically for classification, the input dataset is divided into two samples: training and testing. During Stages 1 and 2, training is performed using the training set, whereas during Stage 3, testing is performed using the testing set. For the testing process, the testing data samples are classified into -1 (minus one) class or +1 (plus one) class, based on the sign value of

the function f(x) in equation (3). In this case, if the output of the equation (3) is less than zero (f(x) < 0), then the test vectors are assigned to -1 class; and if the output of the equation (3) is greater than zero (f(x) > 0), and then the test vectors are assigned to +1 class. We utilize the following formula (derived from [27]), which is known as the decision function (equation (22)), to design and develop our testing (or classification) stage of the CO-based SVM algorithm.

$$\emptyset(x_t) = sgn\{\sum_{i=1}^{m} \alpha_i y_i K(x_i, x_t) + b\} \quad (22)$$

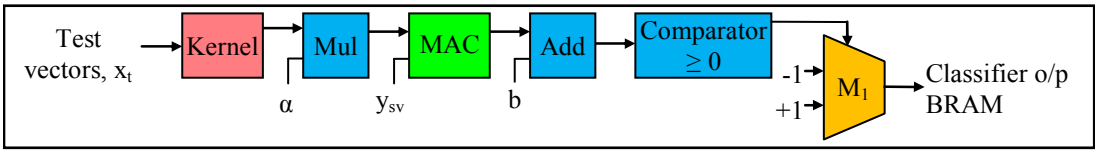


Figure 3.10: Datapath of Testing Process

The datapath of the testing process, which performs the final SVM classification, is illustrated in Figure 3.10. As mentioned before, this datapath is designed and developed based on the equation (22). In this stage, as in equation (22), it is necessary to perform the sign verification for the testing (classification) process. As depicted in Figure 3.10, the datapath for the testing (classification) comprises a multiplier, a MAC module, an adder, a comparator and a multiplexer. Initially, the test vectors (from the testing sample) are pre-fetched to the BRAM from the DDR3-SDRAM and forwarded to the kernel block in a pipelined fashion. As detailed in Stage 1 (Section 3.3), the Kernel module maps these test vectors to the feature space. The two aforementioned steps (i.e., pre-fetching the test vectors, and mapping them to the feature space) are done in stage 1. In Stage 3 of our design, we reuse the pre-fetch and kernel modules from Stage 1 to reduce the total area occupied by our hardware design. The resource utilization is detailed in Section 3.4.

In stage 3, as shown in Figure 3.10, the result of the kernel computation, which is the kernel matrix, is initially multiplied by the α value (obtained from Stage 2) using the Mul module. Then the result of the Mul module is forwarded to the MAC module to multiply with the output label of the support vector (y_{sv}), and then perform the summation operation corresponding to equation (22). The α value and the support vector parameters (y_{sv}) are computed during the convex optimization process, as detailed in Section 3.3. Next, the bias value (b) (obtained in Stage 2 (in equations (2) and (21)) is added (using the Add module) to the final summation result of the MAC module. Then the result of the adder is forwarded to the comparator to determine whether the adder result is greater than or equal to zero. Based on the results of the comparator, the test data samples are assigned into +1 class or -1 class, via the multiplexer.

3.5 EXPERIMENTAL RESULTS AND ANALYSIS

We perform experiments to evaluate the feasibility and efficiency of our proposed embedded designs, for convex optimization (CO) based SVM algorithm, in terms of the speed-performance (speedup), accuracy, as well as the scalability to handle different datasets with varying sizes. We measure the classification accuracy [132] and the speed-performance utilizing the following equations (23) and (24), respectively. The scalability metric is to demonstrate our embedded designs' capability to handle different datasets with varying data sizes, and varying number of attributes and other varying parameters that are commonly found in many datasets of machine learning applications.

$$\text{Accuracy (in \%)}, (y_i, \hat{y}) \quad = \frac{1}{n_i} \sum_{i=0}^{n_i - 1} 1 \ (y_i == \hat{y}) * 100 \tag{23}$$

$$\text{Speedup} = \frac{\text{Software execution time}}{\text{Hardware execution time}} \tag{24}$$

During the initial design phase, we compare our results with the results from an open-source python code [132], in order to verify the correctness and the functionality of our proposed embedded designs. The execution time for the embedded designs are obtained in clock cycles and converted to seconds. Our proposed embedded architectures (both the hardware and software) are executed on Virtex-6 FPGA running at 100MHz, whereas the python code is executed on the desktop computer, with Intel-i7 processor, running at 2.3GHz.

In this research work, the experiments are carried out to evaluate our embedded designs on two different benchmark datasets: Ionosphere dataset [122] and Wisconsin breast cancer diagnostic dataset [21] for machine learning applications. These datasets are stored in the DDR3-SDRAM and formatted accordingly to distinguish between the input features and the output labels. The data size is measured by considering the number of input vectors/samples (n) and the number of dimensions/features (m) in each vector. For our experiments, data sizes are varied to examine its impact on the accuracy, speedup, and scalability.

For all our experiments, we partition the datasets into two sets: training and testing. The test set is considered as a percentage of the dataset to investigate the classification accuracy. In this case, the training set is varied from 10% to 90%, with an increment of 10%.

Apart from varying the data sizes for the testing and training, the number of iterations (to find the minima) is also varied for the training process, in order to examine the ability and the speedup of the convex optimization process to find these minima values.

3.5.1 Analysis on Resource Utilization

In order to examine the feasibility and area-efficiency of our embedded hardware architectures, cost analysis on space (resource utilization) is carried out. In this case, after the implementation process, we obtain the significant resource utilization parameters, including the number of occupied slices, number of BRAMs, and number of DSP48E1 slices, whereas the number of occupied slices typically consist of the slice registers and slice LUTs. These resource utilization statistics for our proposed embedded hardware design is presented in Table 1. As illustrated, for our embedded hardware design, the total number of occupied slices, number of DSP48E1, and number of BRAMs are 5216, 110, and 118 respectively. Considering the total number of logic slices (37680 slices) in Virtex-6 FPGA, our hardware design occupies only 12.7% of the chip area.

Table 1. Resource Utilization for Embedded Hardware.

Description	Occupied area on chip
Number of occupied slices	5216
Number of BRAM 36E1	118
Number of DSP48E1	110
Number of slice registers	12784
Number of slice LUTs	12965

During our initial design phase, we explore the feasibility and tradeoff of utilizing the registers versus BRAMs to store the intermediate results. From this investigation, it is observed that utilization of BRAMs leads to substantial reduction of the total number of occupied slices on the chip compared that of the registers. Furthermore, BRAMs are imperative to hold the intermediate minima values during the sequential minimal optimization (SMO) process. For certain operations, we utilize the DSP48E1 slices for single-precision floating-point computations [135]. This design decision also leads to

more area-efficient and lower clock latency for the floating-point operations compared to ones using the pure logic-based options [136].

Table 2. Execution Time, Speedup, Accuracy for Cancer Benchmark Dataset for Linear Kernel, with C=1, d=2, γ =0.0001.

Data size	Training set (%)	No. of vectors	Micro-Blaze execution time (clock cycles)	Hardware execution time (clock cycles)	Speedup	Accuracy (%)
1707	10	57	161890000	6933463	23.35	83.43
3414	20	114	328230000	13357865	24.57	92.32
5121	30	171	655110000	24470082	26.77	90.47
6828	40	228	1342490000	32207844	41.68	91.52
8535	50	285	2112080000	35770689	59.04	92.28
10242	60	342	3109080000	43691507	71.16	91.66
11949	70	399	4294380000	61637143	69.67	92.39
13656	80	456	5773890000	84519878	68.31	91.22
15363	90	513	7551280000	101299406	74.54	92.98

Table 3. Execution Time, Speedup, Accuracy for Cancer Benchmark Dataset for Polynomial Kernel, with C=1, d=2, γ =0.0001.

Data size	Training set (%)	No. of vectors	Micro-Blaze execution time (clock cycles)	Hardware execution time (clock cycles)	Speedup	Accuracy (%)
1707	10	57	359265478	122811272	2.93	81.09
3414	20	114	1698749510	134533462	12.63	86.18
5121	30	171	4548668857	155902214	29.18	93.48
6828	40	228	7772314965	184562787	42.11	88.88
8535	50	285	12353625978	222771368	55.45	86.31
10242	60	342	19230324150	268144802	71.72	93.42
11949	70	399	19744400000	323179673	61.09	92.98
13656	80	456	25290974839	385245965	65.65	92.98
15363	90	513	27795974670	460237826	60.39	96.49

Utilizing the on-chip BRAMs (and in few cases, using the registers) to hold the intermediate results, substantially reduces the execution time for numerous matrix computations inherent in the CO-based SVM algorithm, thus enhancing the overall speed-performance of this algorithm as illustrated in Section 5.4.

The pre-fetching techniques introduced to reduce the memory access latency and the on-chip BRAM to hold the data/results, indeed add more space (i.e., extra resources) to the overall design of the CO-based SVM algorithm. Thus, it is important to consider the speed-space tradeoffs, when designing certain algorithms/techniques, such as CO-based SVM, for machine learning applications, which typically require processing large volume of data, especially on embedded platforms with their stringent area constraints.

3.5.2 Analysis on Classification Accuracy

We perform experiments to evaluate the classification accuracy of our proposed embedded designs for the CO-based SVM algorithm. In this case, the classification accuracy for the CO-based SVM algorithm is obtained with the varying data sizes for the maximum number of iteration of 1000. The classification accuracy is measured using equation (23).

In order to measure the classification accuracy, we partition the datasets into two sets: training and testing. The training set is varied from 10% to 90%, with an increment of 10% to investigate the classification accuracy. Furthermore, for the linear, polynomial, and Gaussian radial basis function (RBF) in Stage 1, we investigate and select the following specifications: penalty parameter (C) to 1; degree of polynomial (d) to 2, with coefficient of 1; and γ to 0.0001.

The aforementioned parameters are varied to find a good fit for constructing the hyper-plane. Varying these parameters can potentially lead to under-fitting and over-fitting problems [132]. The under-fitting occurs when the SVM generalizes the main features of the data; whereas the over-fitting occurs when the SVM learns that it is sensitive to the noise [27], [132]. As a result, for our experiments, we partition the

datasets and utilize the cross-validation method [27] to select suitable constants. The cross-validation methods enable us to train the SVM by partitioning the dataset, and also enable us to adjust the aforementioned parameters to obtain the best accuracy results. In this case, in order to avoid the under-fitting and over-fitting issues, the SVM is trained the tested with different parameters (C, degree, gamma, number of iterations) to construct a better data classifier.

Table 4. Execution Time, Speedup, Accuracy for Cancer Benchmark Dataset for Gaussian RBF Kernel, with C=1, d=2, γ =0.0001.

Data size	Training set (%)	No. of vectors	Micro-Blaze execution time (clock cycles)	Hardware execution time (clock cycles)	Speedup	Accuracy (%)
1707	10	57	622351344	46237098	13.46	73.54
3414	20	114	1699669752	94583736	17.97	88.79
5121	30	171	2762669586	110772637	24.94	89.61
6828	40	228	4754029339	128835483	36.9	92.72
8535	50	285	7267484089	148984913	48.78	92.66
10242	60	342	10895192675	224134801	48.61	92.48
11949	70	399	15652953272	337420850	46.39	91.14
13656	80	456	22175875454	385131564	57.58	90.81
15363	90	513	25205111592	463926221	54.33	90.27

Table 5. Execution Time, Speedup, Accuracy for Ionosphere Benchmark Dataset for Linear Kernel, with C=1, d=2, γ =0.0001.

Data size	Training set (%)	No. of vectors	Micro-Blaze execution time (clock cycles)	Hardware execution time (clock cycles)	Speedup	Accuracy (%)
1194	10	36	339293000	10589052	32.04	65.94
2387	20	71	1002549999	30265650	33.12	67.53
3581	30	106	2088900000	52608548	39.7	69.78
4774	40	141	3566149999	86932128	41.02	61.66
5967	50	176	5243630000	102803892	51	77.26
7161	60	211	7641730000	134483893	56.82	75.6
8354	70	246	10094000000	162238668	62.21	96.71
9547	80	281	13728500000	218501180	62.83	95.12
10741	90	316	19554900000	311316697	62.81	100

The classification accuracy results for the overall CO-based SVM algorithm using Cancer and Ionosphere benchmark datasets are presented in Tables 2-4 and 5-7, respectively. Three sets of accuracy results (in percentage) are obtained separately, when using three different mathematical kernels for Stage 1, i.e., linear (in Tables 2 and 5), polynomial (in Tables 3 and 6, and Gaussian RBF (in Tables 4 and 7). The accuracy results are presented in column 7 of these tables.

Table 6. Execution Time, Speedup, Accuracy for Ionosphere Benchmark Dataset for Polynomial Kernel, with C=1, d=2, γ =0.0001.

Data size	Training set (%)	No. of vectors	Micro-Blaze execution time (clock cycles)	Hardware execution time (clock cycles)	Speedup	Accuracy (%)
1194	10	36	454131374	127739374	3.55	65.82
2387	20	71	1109957922	124685905	8.9	67.96
3581	30	106	2412208381	132320757	18.23	69.16
4774	40	141	4156918296	147575466	28.16	61.47
5967	50	176	6088204069	163878344	37.15	77.65
7161	60	211	8680224863	182982124	47.43	75.68
8354	70	246	11520789306	206488474	55.79	96.2
9547	80	281	15680125496	239989386	65.33	100
10741	90	316	20949687217	264792097	79.11	100

Table 7. Execution Time, Speedup, Accuracy for Ionosphere Benchmark Dataset for Gaussian RBF Kernel, with C=1, d=2, γ =0.0001.

Data size	Training set (%)	No. of vectors	Micro-Blaze execution time (clock cycles)	Hardware execution time (clock cycles)	Speedup	Accuracy (%)
1194	10	36	131588528	17246202	7.63	60.32
2387	20	71	443339285	33586309	13.2	62.2
3581	30	106	1369622549	62597008	21.88	71.54
4774	40	141	2732821550	102237992	26.73	74.53
5967	50	176	3938128950	136693125	28.81	77.67
7161	60	211	8127266954	241523535	33.65	68.18
8354	70	246	7150427148	191957775	37.25	94.33
9547	80	281	20918761361	528652043	39.57	97.59
10741	90	316	29338152919	718544034	40.83	97.79

From Tables 2 to 7, it is observed that the classification accuracy varies with the different datasets as well as with varying percentage of training sets. At a glance, the classification accuracy seems to increase with the increasing percentage of training set for both the datasets. For instance, classification accuracy increases: from 83%-93% (in Table 2) and from 66%-100% (in Table 5) with the linear kernel; from 81%-96% (in Table 3) and 66%-100% (in Table 6) with the polynomial kernel; and 74%-90% (in Table 4) and 60%-98% (in Table 7) with the Gaussian RBF kernel. From Tables 5 to 7, the Ionosphere datasets has 100% classification accuracy, when the percentage of training set is 90% of the dataset with linear and polynomial kernels. From Table 2 to 4, the Cancer benchmark dataset achieves the best classification accuracy of 96% with the polynomial kernel, when the percentage of training set is 90% of the dataset. It should be noted that the classification accuracy results are the same for our embedded hardware design as well as for our embedded software design.

Apart from our embedded hardware and software designs, the classification accuracy experiments are also performed on the python code running on the desktop computer. The accuracy results when using the linear, polynomial, and Gaussian RBF mathematical kernels are presented in Figures 3.11 and 3.12 for the Cancer and Ionosphere benchmark datasets (with the maximum number of iteration of 1000), respectively. The accuracy results of our designs when using the linear, polynomial, and Gaussian RBF mathematical kernels (from Tables 2 to 7) are also presented in Figures 3.11 and 3.12 for the Cancer and Ionosphere benchmark datasets, respectively.

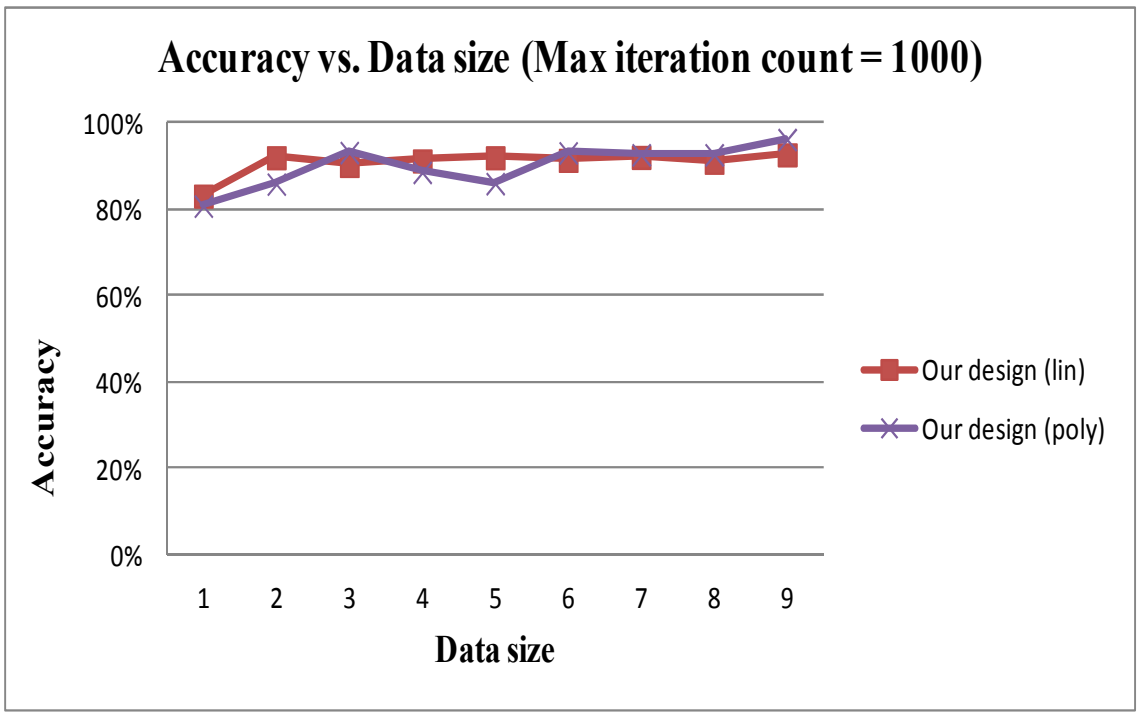


Figure 3.11: Graph of Classification Accuracy vs. Data Size for Cancer Benchmark Dataset

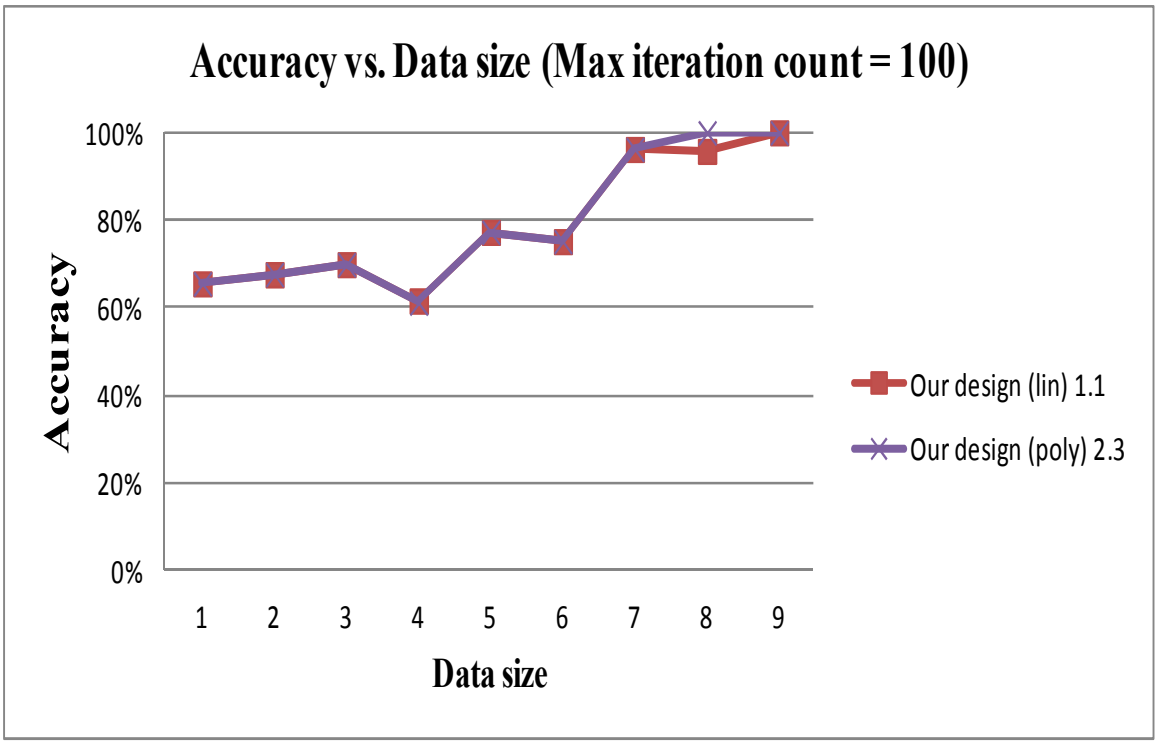


Figure 3.12: Graph of Classification Accuracy vs. Data Size for Ionosphere Benchmark Dataset

From these results, it is evident that classification accuracy varies with different datasets, with varying data sizes, as well as with different classification techniques. As detailed in Chapter 2, selecting a suitable classifier for a specific dataset is not a trivial task. By employing the cross-validation method, we can vary and select the most appropriate parameters that can indeed facilitate this task, which in turn will lead to better classification results.

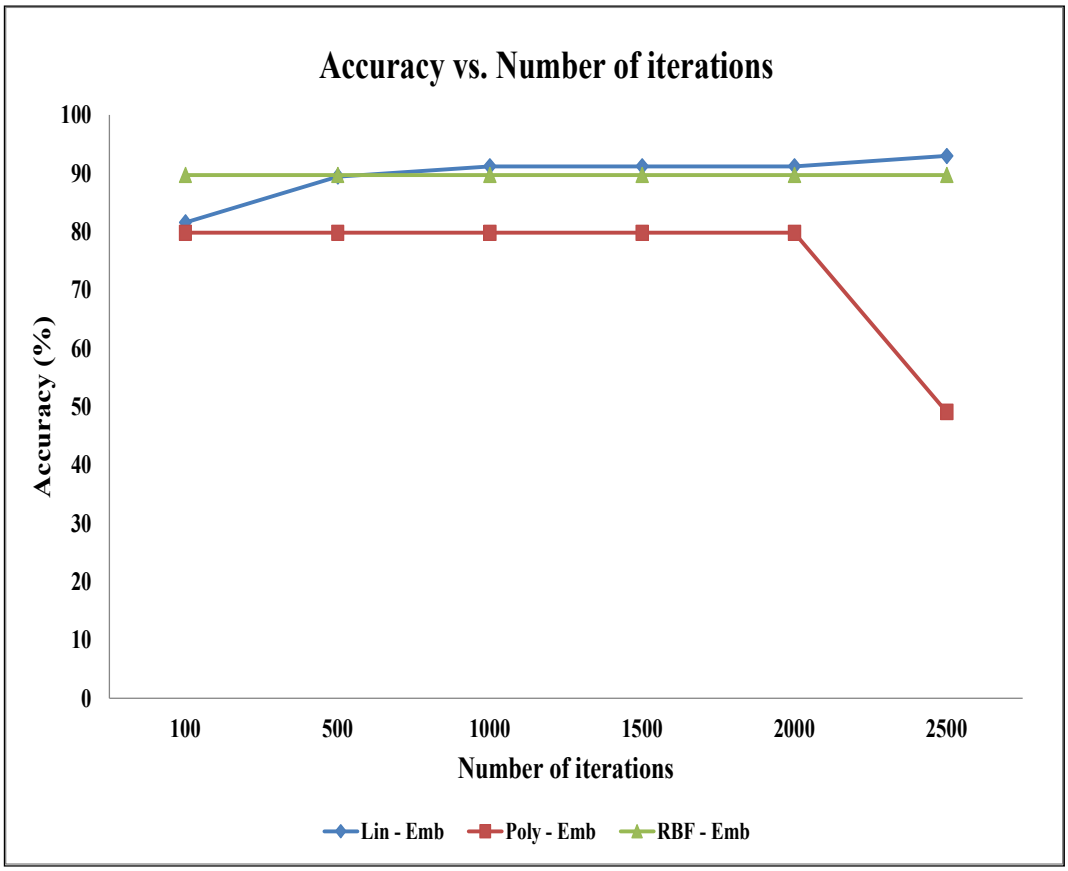


Figure 3.13: Graph of Classification Accuracy vs. Number of Iterations for Cancer Benchmark Dataset

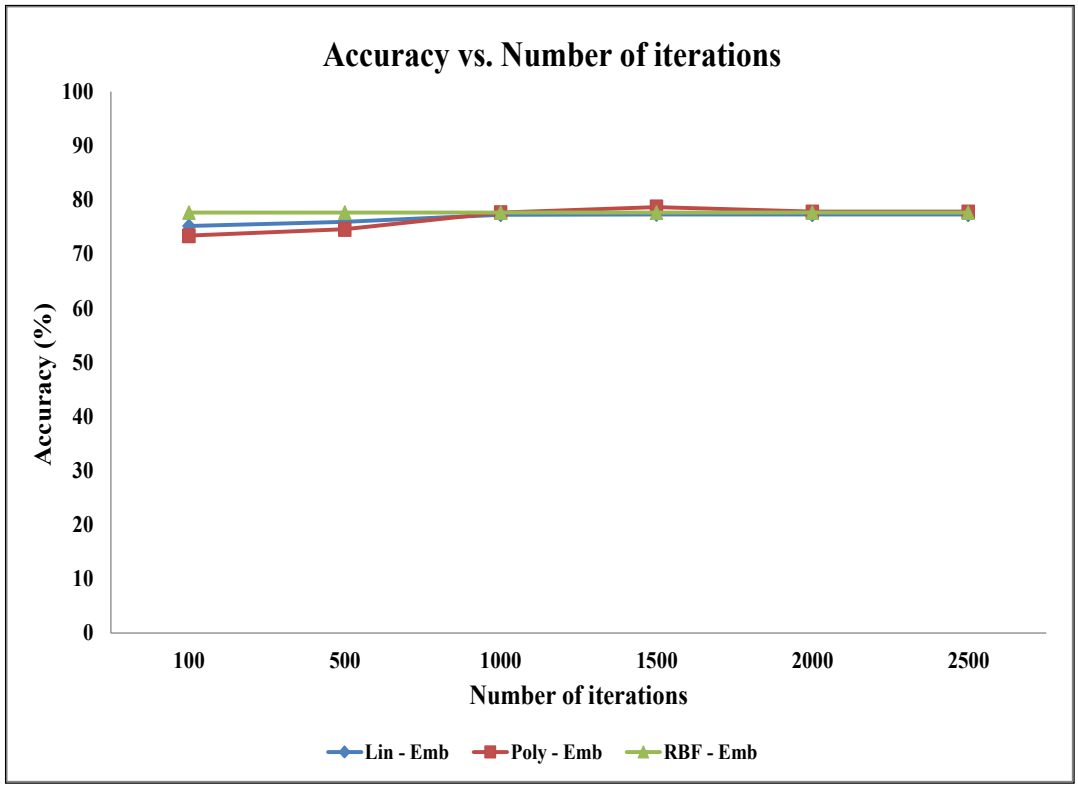


Figure 3.14: Graph of Classification Accuracy vs. Number of Iterations for Ionosphere Benchmark Dataset

3.5.2.1 Analysis of Classification Accuracy with Varying Number of Iterations

The aforementioned classification accuracy results are obtained with varying data sizes and with constant number of iterations. We perform additional experiments to analyze the classification accuracy results with varying number of iterations and with constant data size. In this case, we select the training set data size of 50% for both the Ionosphere and Cancer datasets. In this case, we vary the maximum number of iterations from 100 to 2500 with an increment of 500, to find the minima value. The classification accuracy results of our designs when using the linear, polynomial, and RBF mathematical kernels are shown in Figures 3.13 and 3.14, for the Cancer and Ionosphere benchmark datasets, respectively.

Table 8. Accuracy vs. Number of iterations – Cancer Dataset

# of iterations	Accuracy - Embedded Design		
	Lin - Embedded design	**Poly - Embedded design**	**RBF - Embedded design**
100	81.57	79.82	89.74
500	89.47	79.82	89.74
1000	91.22	79.82	89.74
1500	91.22	79.82	89.74
2000	91.22	79.82	89.74
2500	92.98	49.12	89.74

Note: C = 1, degree (d) = 2, gamma (γ) = 0.0001, Data size = 50%

Table 9. Accuracy vs. Number of iterations – Ionosphere Dataset

# of iterations	Accuracy - Embedded Design		
	Lin - Embedded design	**Poly - Embedded design**	**RBF - Embedded design**
100	81.57	79.82	89.74
500	89.47	79.82	89.74
1000	91.22	79.82	89.74
1500	91.22	79.82	89.74
2000	91.22	79.82	89.74
2500	92.98	49.12	89.74

Note: C = 1, degree (d) = 2, gamma (γ) = 0.0001, Data size = 50%

For the Ionosphere datasets, it is observed that the convex optimization process converges to the minima with less number of iterations, for instance in 100 iterations in some cases. As a result, the impact of number of iterations on accuracy is insignificant, as shown in Figure 3.14.

3.5.3 Analysis on Execution Time

As detailed in Section 3.2, in order to evaluate the speed-performance of our embedded hardware designs, we design and implement the embedded software for the CO-based SVM algorithm. The software design is executed on the Micro-Blaze soft processor on the same ML605 development platform. The execution times for both the embedded hardware and software designs are obtained using the AXI Timer running at 100MHz on the ML605 board. These execution times are measured in real-time, while our designs are actually running on the chip. In this case, we design the AXI Timer in cascade mode to measure the accurate execution time for all three stages of the CO-based SVM algorithm. This is mainly because in certain scenarios, especially for large datasets, the execution time exceeds the allowable timer counter value of the AXI Timer. In order to resolve the counter overflow issue, the AXI timer is designed utilizing two timers in cascade mode.

The execution times for both our embedded hardware and software designs are obtained with the varying data sizes for the maximum number of iterations of 1000. The execution times for the overall CO-based SVM algorithm using the Cancer and Ionosphere benchmark datasets are presented in Tables 2-4 and 5-7, respectively. Similar to the classification accuracy results, three sets of execution times for embedded software and embedded hardware designs are obtained separately, when using three different mathematical kernels for Stage 1, i.e., linear (in Tables 2 and 5), polynomial (in Tables 3 and 6), and RBF (in Tables 4 and 7). The execution time for each set (for both the embedded hardware and software) is measured 10 times and the average is presented in columns 4 and 5 of these tables, respectively.

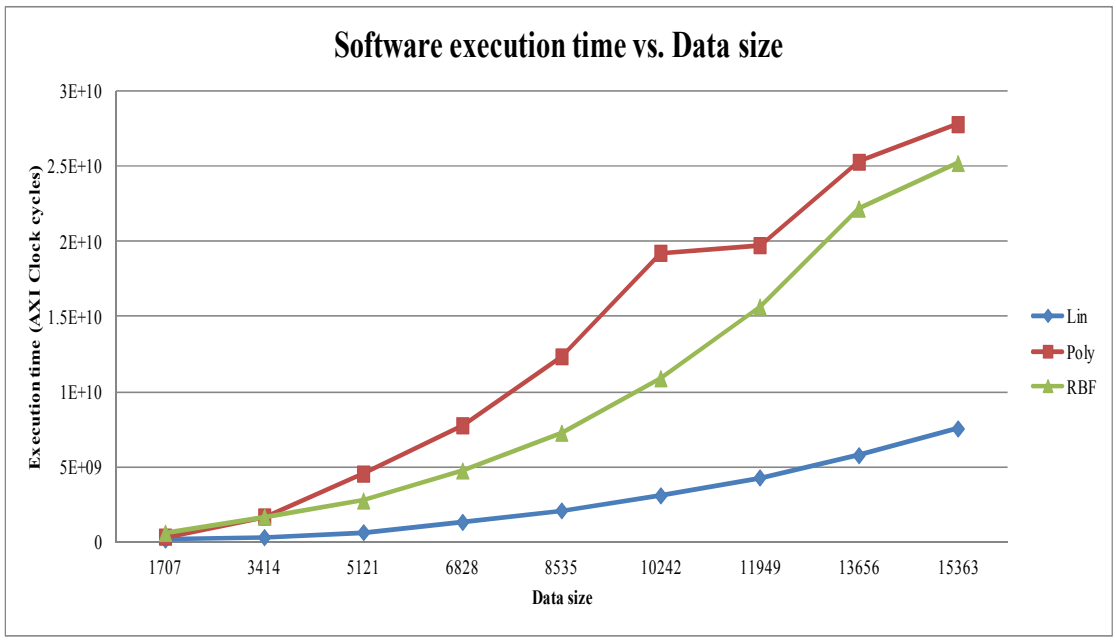


Figure 3.15: Embedded Software for CO-Based SVM: Execution Times vs. Data Size for Cancer Benchmark Dataset

The execution times for the embedded hardware and software designs for the CO-based SVM algorithm using the linear, polynomial, and Gaussian RBF kernels are illustrated in Figure 3.15 and 3.16, respectively, for the Cancer benchmark datasets. As illustrated, the execution times increase almost exponentially with the increasing data sizes, for both the embedded hardware and software designs. Somewhat similar results are obtained when using the Ionosphere dataset. The CO-based SVM with linear kernel takes less execution time compared to that of the polynomial kernel. As illustrated in Figure 3.15 and 3.16, the execution times are the highest for CO-based SVM with the polynomial kernels, whereas the execution times are the lowest for CO-based SVM with the linear kernels for the Cancer dataset.

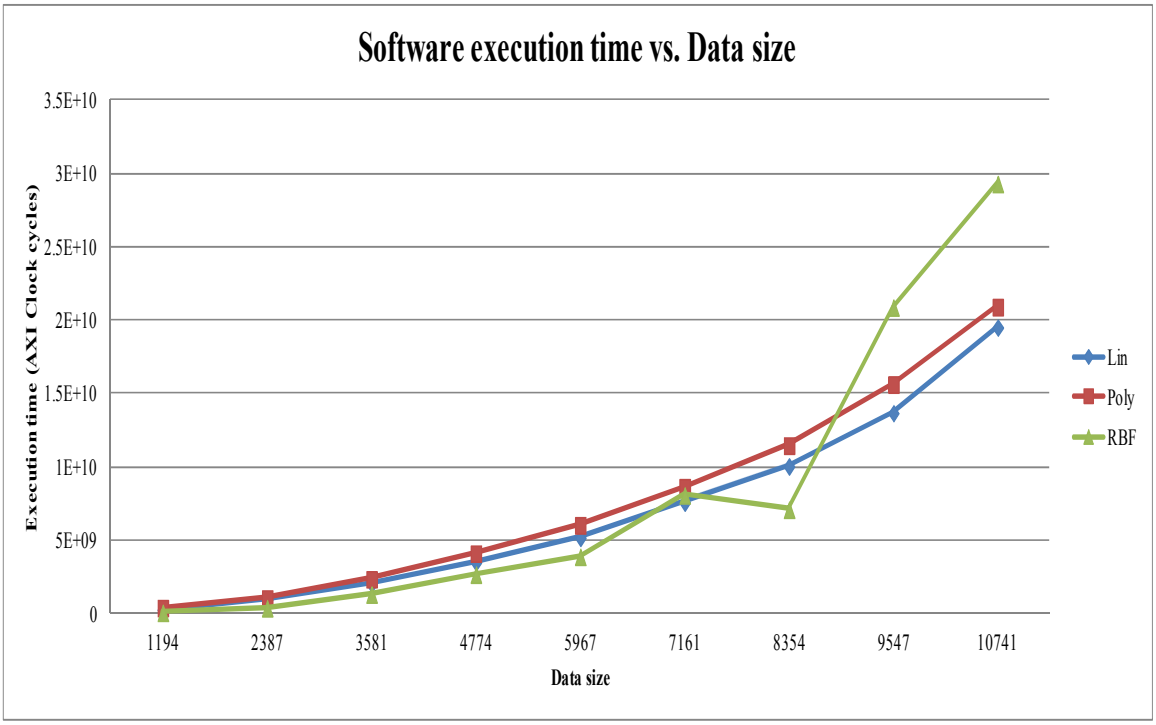


Figure 3.15.1: Embedded Software for CO-Based SVM: Execution Times vs. Data Size for Ionosphere Benchmark Dataset.

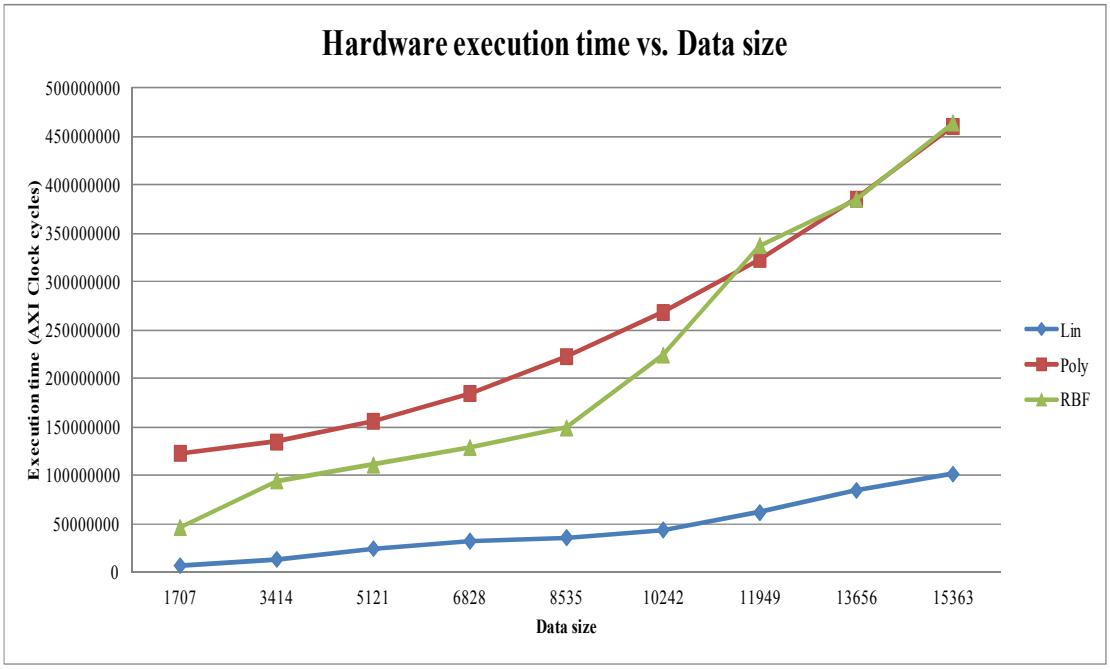


Figure 3.16: Embedded Hardware for CO-Based SVM: Execution Times vs. Data Size for Cancer Benchmark Dataset

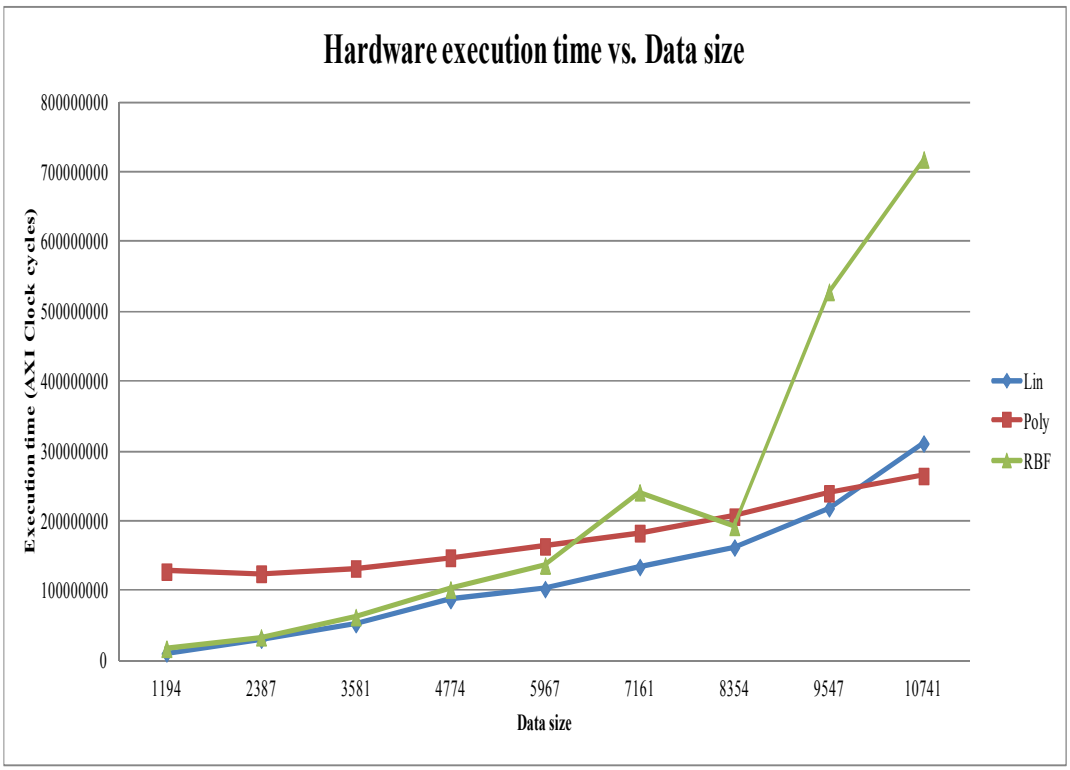


Figure 3.16.1: Embedded Hardware for CO-Based SVM: Execution Times vs. Data Size for Ionosphere Benchmark Dataset.

Table 10. Hardware execution time vs. Number of iterations – Cancer Dataset

Max. iterations	Linear	Polynomial	RBF
500	47881098	57628552	88430257
1000	35770689	52771368	148984913
1500	80857778	102599012	152423593
2000	128725569	126131230	162257694
2500	150962104	148160823	175812280
3000	143920135	193942670	178098186

Note: C = 1, degree (d) = 2, gamma (γ) = 0.0001, Data size = 50%

3.5.3.1 Analysis of Execution Times with Varying Number of Iterations

The aforementioned execution times are obtained with varying data sizes and with constant number of iterations. Similar to accuracy analysis, we perform additional

experiments to analyze the execution times with varying number of iterations and with constant data size. In this case, we vary the maximum number of iterations from 500 to 3000 with an increment of 500, to find the minima value as shown in Table 10 and 11.

Table 11. Hardware execution time vs. Number of iterations – Ionosphere Dataset

Max. iterations	Linear	Polynomial	RBF
500	111498241	75833237	136693567
1000	102803892	163878344	136693125
1500	204012481	214907978	136693183
2000	278499821	298532129	136692360
2500	278858795	312702454	136693468
3000	304813706	362067934	136693340

Note: C = 1, degree (d) = 2, gamma (γ) = 0.0001, Data size = 50%

The execution times for both the embedded hardware and software are also obtained with the varying number of iterations for both the benchmark datasets with the training set data size of 50%. The embedded hardware execution times for Cancer and Ionosphere datasets are presented in Figures 3.17 and 3.18, respectively. Visually, as shown in Figure 3.17, for our embedded hardware designs, the execution times increase almost linearly with the increasing number of iterations, for all three kernels, for the Cancer benchmark datasets. For the Ionosphere benchmark dataset, as depicted in Figure 3.18, the embedded hardware execution times increase almost linearly with the increasing number of iterations, for the linear and polynomial kernels; whereas for the RBF kernel, the embedded hardware execution times remain the same with the increasing number of iterations.

For our designs, we utilize the maximum number of iteration (in our case, 1000 iterations) as our threshold point to find the minima value, instead of implementing a

specific stopping criterion. Hence, our convex optimization solver has to reach the maximum number of iterations, in order to complete the execution of the CO-based SVM algorithm. Conversely, a stopping criterion terminates the execution of the CO-based SVM algorithm, when the objective function converges to the minima value, which may or may not reduce (or increase) the total execution time.

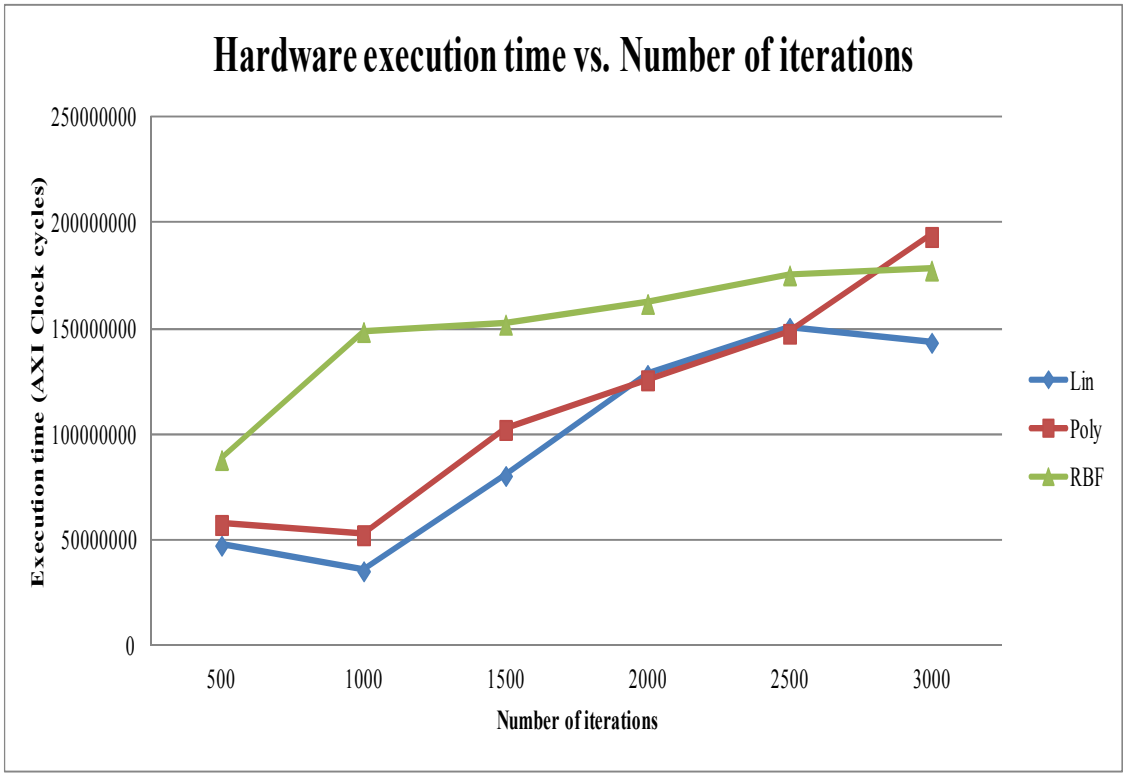


Figure 3.17: Embedded Hardware for CO-Based SVM: Execution Times vs. Number of Iterations for Cancer Benchmark Dataset

3.5.4 Analysis on Speedup

The performance-gain (or speedup), resulting from the embedded hardware design over embedded software running on Micro-Blaze, for the CO-based SVM algorithm using three different mathematical kernels, is presented in column 6 in Tables 2-7. The speedup is measured using equation (24). Figures 3.19 and 3.20 demonstrate the

speedup versus the data sizes (percentage of training set) for our embedded hardware design for the CO-based SVM with the linear, polynomial, and Gaussian RBF kernels for the Cancer and Ionosphere benchmark datasets, respectively. At a glance, as shown in Figures 3.19 and 3.20, the speedup typically increases as the percentage of training set increases for both the datasets. Also, Table 12 and 13 illustrates the speedup different computing platforms with respect to hardware design. In Table 12 and 13, the execution time is measured in seconds the speedup is computed by dividing the chosen platform execution time over respective platform. The result is the speedup comparison among different available computing platforms.

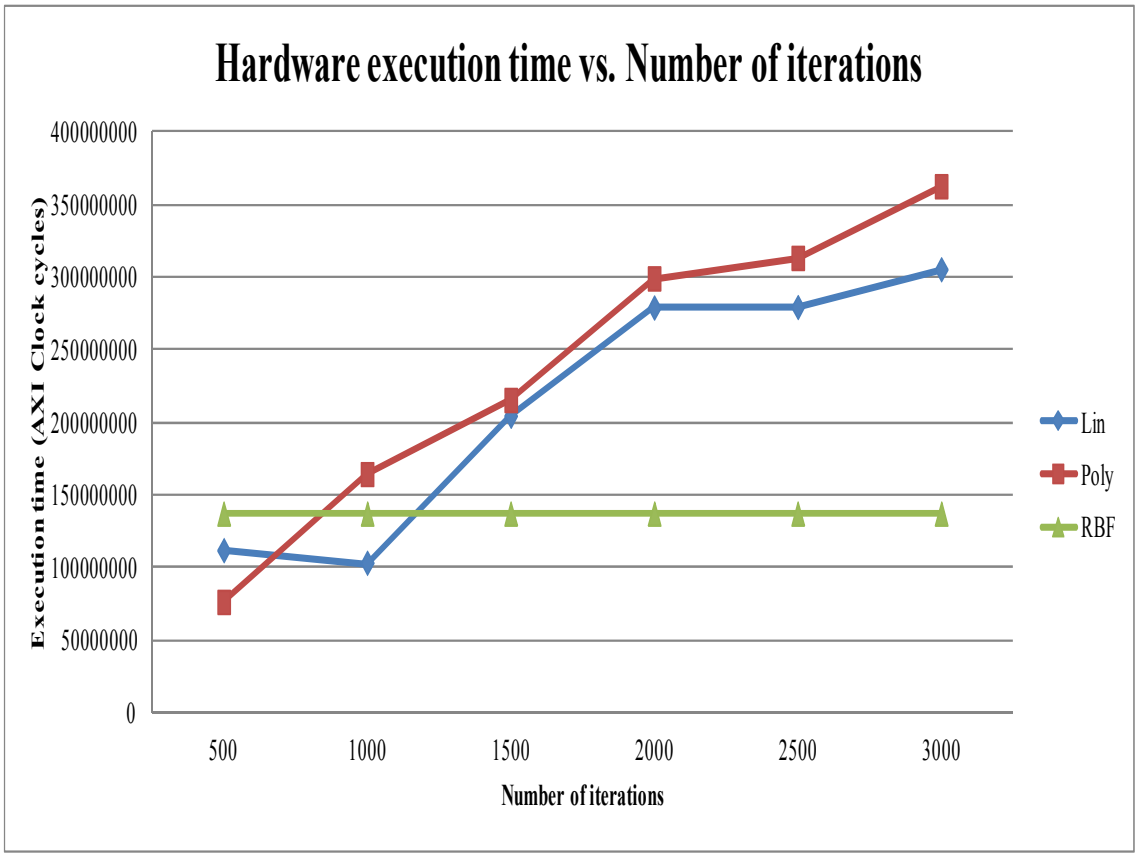


Figure 3.18: Embedded Hardware for CO-Based SVM: Execution Times vs. Number of Iterations for Ionosphere Benchmark Dataset

Table 12. Speedup Comparison – Cancer Dataset

Platform	Execution Time (s)			Speedup over SW			Speedup over python		
	Lin	Poly	RBF	Linear	Poly	RBF	Lin	Poly	RBF
SW-100MHz	75.51	277.96	252.05	-	-	-	0.04	0.01	0.004
Python-2.3GHz	3.16	3.0319	1.0156	23.84	91.67	248.17	-	-	-
HW-100MHz	1.01	4.6023	4.6392	74.54	60.39	54.3307	3.12	0.65	0.21

Note: C = 1, degree (d) = 2, gamma (γ) = 0.0001, maximum number of iterations = 1000

Table 13. Speedup Comparison – Ionosphere dataset

Platform	Execution Time (s)			Speedup over SW			Speedup over python		
	Lin	Poly	RBF	Lin	Poly	RBF	Lin	Poly	RBF
SW-100MHz	195.54	209.49	293.38	-	-	-	0.01	0.43	0.001
Python-2.3GHz	3.76	92.02	0.46	51.93	2.27	626.88	-	-	-
HW-100MHz	3.11	2.64	7.18	62.81	79.11	40.83	1.20	34.75	0.06

Note: C = 1, degree (d) = 2, gamma (γ) = 0.0001, maximum number of iterations = 1000

For the Ionosphere benchmark dataset, as in Figure 3.20, for the one with polynomial kernel, the speedup increases linearly when the percentage of training set increases from 10% to 100%; for the one with linear kernel, the speedup increases almost linearly when the percentage of training set increases from 10% to 80%, and the speedup remains the same when the percentage of training set increases from 80% to 100%; for the one with RBF kernel also the speedup increases almost linearly when the percentage of training set increases from 10% to 100%.

For the Cancer and Ionosphere benchmark dataset, as in Figure 3.19 and 3.20, for the one with polynomial kernel, the speedup increases linearly when the percentage of training set increases from 10% to 70%, and the speedup drops slightly when the percentage of training set increases from 70% to 100%; for the one with linear kernel, the

speedup increases linearly when the percentage of training set increases from 10% to 70 and from 90% to 100; for the one with RBF, the speedup increases linearly when the percentage of training set increases from 10% to 60% and from 80% to 90% and the speedup drops slightly when the when the percentage of training set increases from 60% to 80% and from 90% to 100%.

Figure 3.19: Embedded Hardware for CO-Based SVM: Speedup vs. Data Size for Cancer Benchmark Dataset

To provide a comparison with the available source, Table 14 ad 15 provides the accuracy results with the similar test data size. It is evident from Table 4-7 and Table 14-15 that for the same test environment, the results are similar. We can further evaluate with more complex patterns to illustrate the effects of SVM parameters, size of data sets etc for further analysis.

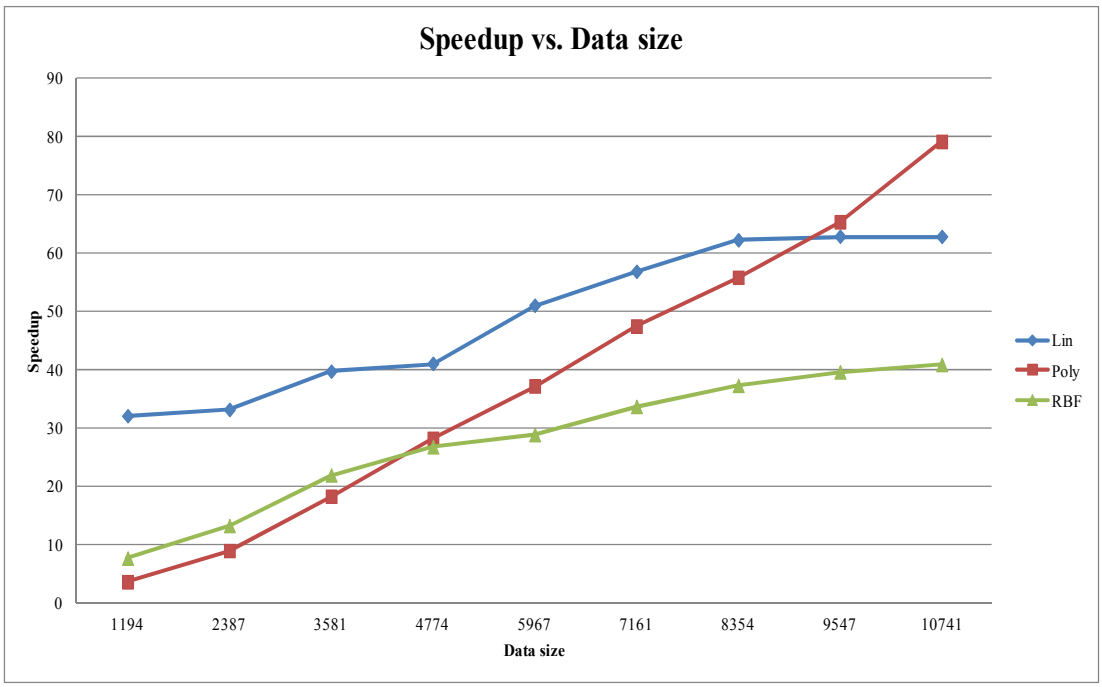


Figure 3.20: Embedded Hardware for CO-Based SVM: Speedup vs. Data Size for Ionosphere Benchmark Dataset

Table 14. Python code - Accuracy for Cancer Benchmark Dataset

			Accuracy (%)		
Data size	**Training set (%)**	**# vectors**	**Lin - Python Design**	**Poly - Python Design**	**RBF - Python Design**
1707	10	57	82.58	87.08	68.1
414	20	114	70.48	94.93	87
5121	30	171	95.46	66.75	92.44
6828	40	228	75.88	91.17	92.94
8535	50	285	81.62	55.83	91.87
10242	60	342	82.74	96.46	93.36
11949	70	399	96.44	95.26	93.49
13656	80	456	25	61.6	93.75
15363	90	513	10.9	16.36	96.36

Note: C = 1, degree (d) = 2, gamma (γ) = 0.0001, Maximum number of iterations = 1000

Table 15. Python code - Accuracy for Ionosphere Benchmark Dataset

			Accuracy (%)		
Data size	Training set (%)	# vectors	Lin - Python Design	Poly - Python Design	RBF - Python Design
1194	10	36	82.8	69.74	66.56
2387	20	71	81.36	73.47	67.74
3581	30	106	82.37	68.85	70.08
4774	40	141	84.68	80.86	73.2
5967	50	176	86.2	91.37	78.16
7161	60	211	94.24	90.64	68.34
8354	70	246	99.03	95.19	95.19
9547	80	281	94.2	97.1	97.1
10741	90	316	91.17	97.05	97.05

Note: C = 1, degree (d) = 2, gamma (γ) = 0.0001, Maximum number of iterations = 1000

As mentioned earlier, additional software experiments are performed on a desktop computer; thus, we also compare our embedded hardware designs running at 100MHz on Virtex-6 FPGA with the baseline python software design on the Intel i7 processor running at 2.3GHz. In this case, our embedded hardware design achieves 3.1 times speedup compared to the python design for the Cancer dataset with the linear kernel; and our embedded hardware design achieves 34.8 times speedup compared to the python design for the Ionosphere dataset with the polynomial kernel.

In summary: It is observed that for the CO-based SVM algorithm, as the number of samples (i.e., vectors) increases, the accuracy and total speedup also increase. In this case, when the CO-based SVM classifier has more samples to learn, it could lead to identifying complex patterns, and also generating a better separating hyper-plane. Furthermore, as the size of the matrices is increasing, as well as the complexity of the

computations/operations is increasing, customized and optimized designs might be the best avenue to accelerate and enhance various performance metrics of the CO-based SVM algorithms, compared to the conventional computing platforms such as general-purpose processors.

3.5.5 Analysis on Existing Works on FPGA-Based Hardware Architectures for CO-Based SVM

We performed an extensive investigation on the existing works on FPGA-based hardware architectures for CO-based SVM algorithms in the published literature. Since we could not find any related work specifically for CO-based SVM, we extended our investigations to the existing works on FPGA-based hardware for general SVM. Our investigation revealed that there are many papers on FPGA-based hardware for SVM; however, we decided to select, discuss, and present some papers that are most recent and/or closely related to our proposed hardware architectures and techniques for creating CO-based SVM. Hence, it should be noted that this is not an exhaustive analysis on the existing works on FPGA-based hardware architectures for SVM. Detailed analysis of other existing works can be found in some survey papers such as [33], [34].

FPGA-based parallel processing hardware architecture was proposed for SVM using stochastic gradient descent (SGD) as the training method, in [35]. The authors demonstrated the scalability of the SGD approach for SVM in terms of fixed-point vs. single-precision floating-point computations. The hardware design was generated using the Xilinx System Generator design tool, and executed on Xilinx ML605 board with Virtex-6 FPGA. In this case, the synthesis results were obtained and reported, in terms of area, time, and throughput; however, the classification accuracy results were not reported. From the results, it is evident that parallelization led to the increase in occupied area, thus

confirming that higher speedup due to parallelization, comes with the penalty of larger occupied area on chip. The proposed design could have demonstrated to execute datasets with more than 4 features/attributes, which is indeed a limitation when executing large volume of data with many attributes. Conversely, our proposed design can execute datasets with varying sizes and with any number of features/attributes.

In [4], an energy-efficient embedded binarized SVM architecture was proposed and implemented on an FPGA. The computation kernels were designed in C/C++ and transformed into HDL using Xilinx HLS (high-level synthesis) tools. The proposed hardware design was executed on Xilinx Virtex-6/7 FPGAs. The results were obtained and reported, in terms of area, speedup, power, and classification accuracy. The FPGA's performance matric results (especially speedup and power) were compared with that of the CPU and GPU. From the results, it is evident that FPGA and GPU achieved significant speedup compared to the CPU. The power consumption of the GPU was significantly higher than that of the FPGA. These results illustrate that FPGA-based hardware architecture for SVM can achieve better performance-per-Watt, thus suitable for embedded devices with stringent power requirements.

An FPGA-based hardware accelerator was proposed for approximate SVM in [36], utilizing two approximation techniques, including precision scaling and loop perforation. The hardware was designed using Xilinx Vivado HLS tool, and executed on Xilinx Zynq7 ZC706 board. The results were obtained and reported, in terms of area, speedup, and classification accuracy. From the results, it is evident that the approximate computing led to higher speedup, but with the penalty of larger occupied area (or

resource utilization) on chip, and lower classification accuracy. In some cases, the significant accuracy loss did not compensate with significant increase in speedup.

In [37], an FPGA-based hardware design was proposed for SVM classifier. In this case, three variable-size SVM models were implemented using different optimization techniques. The proposed hardware was designed using Xilinx Vivado HLS tool, and executed on Xilinx Zynq7 ZC702 board. The results were obtained and reported, in terms of area, speedup, power, and classification accuracy. Also, in this paper, the training phase was done offline on software; hence, the support vectors were pre-computed, and forwarded to the proposed hardware design, which is created only for the testing phase.

An FPGA-based parallel processing architecture was proposed in [38] for training phase of SVM using Sequential Minimal Optimization (SMO). The proposed hardware design was executed on Xilinx Virtex-6/7/Ultra-scale FPGAs. The synthesis results were obtained and reported, in terms of area, throughput, and speedup; however, the classification accuracy results were not reported. In this case, the authors utilized the hardware friendly kernel (HFK) for SVM training, which leads to reduction in precision of the floating-point operations. Although marginal loss in accuracy is acceptable for testing, utilizing HFKs for training would result in an inefficient construction of a hyper-plane during training.

In [39], a FPGA-based hardware-software co-design was proposed to accelerate the SVM algorithm by utilizing a two-level approach: first to optimize the global structure of the SVM; and second to refine it through the design exploration. The proposed architecture was designed using Xilinx Vivado HLS tool, and executed on Xilinx Zynq Zedboard. The results were obtained and reported, in terms of area, latency,

and speedup; however, the classification accuracy results were not reported. As authors indicated, for high values of SVM parameters, the resource utilization (i.e., occupied area) increased significantly, which would be an issue for embedded devices with stringent area requirements. In this paper, the authors extensively discuss and analyze the advantages/disadvantages of utilizing the HLS tools to transform the designs written in C/C+ to HDL, thus providing insight into the HLS inefficiencies, which would be very useful when creating optimized hardware architectures in order to improve certain performance metrics, including the latency.

An FPGA-based coarse-grained reconfigurable hardware architecture was proposed in [40], for various machine learning (ML) algorithms, including SVM, decision trees, and artificial neural networks. The hardware was designed using Xilinx Vivado tool, and executed on Xilinx Virtex-7 FPGA. The results were obtained and reported, in terms of area, and speedup; however, the classification accuracy results were not reported. In this case, in order to change from one ML algorithm to another, authors claim that the reconfigurable processing nodes (RPNs) of the proposed architecture, can be reconfigured individually; however, no details are provided how this can be done. This requires partial reconfiguration of the FPGA; thus, adding significant complexity to the design process, which has not been addressed or discussed in the paper.

A scalable FPGA-based architecture was proposed in [41] to accelerate the SVM classification. The hardware was designed in VHDL, and executed on Altera Stratix III EP3SE260 board. The results were obtained and reported, in terms of speedup; however, the occupied area was not reported. Furthermore, in this paper, the authors only proposed the hardware design for the testing phase. Hence, the support vectors were pre-computed

and stored in the on-chip memory for subsequent processing during the testing phase. The same authors proposed a design flow for the SVM training phase in [42].

From this investigation, it is evident that most of the existing works proposed hardware architectures either for testing or for training, but not for both. In summary, from this investigation, and to the best of our knowledge, we could not find any similar work as ours, in the published literature, that provides FPGA-based hardware accelerators for CO-based SVM, especially on embedded devices, nor could we find any similar work that proposed system-level architectures, which is imperative for the machine learning applications in real-world scenarios.

3.6 Chapter Summary

In this chapter, we introduced novel, unique, customized, and optimized FPGA-based hardware accelerator for convex optimization (CO)-based support vectors machines (SVM) on embedded devices. Our hardware architectures for CO-based SVM were created in such a way to be generic, parameterized, and scalable; hence, without changing the internal architectures, our hardware designs can be used to process different datasets with varying sizes; can be executed on different embedded platforms, including the platforms with recent FPGAs such as Virtex-7 chips; and can be utilized for linear/non-linear separable, multi-dimensional datasets, making it suitable for various machine learning applications such as medical testing for cancer diagnosis, data analysis for quality control, image classifications, and speech recognition. By providing generic and independent IPs (intellectual properties) for each stage, these independent IPs can be utilized for any machine learning or similar application, and are not limited to a specific application.

Our proposed embedded hardware accelerators can be reconfigured (on-the-fly) to select the most suitable mathematical kernel (out of three different kernels, i.e., linear, polynomial, and Gaussian radial basis function), based on the requirements of a given machine learning application. Our proposed hardware architectures/accelerators and techniques were created and optimized considering the constraints associated with the embedded devices as well as the requirements of the CO-based SVM algorithm.

We also introduced unique and efficient system-level architecture for our proposed embedded hardware accelerators for CO-based SVM, in order to process the data efficiently and effectively. With the system-level architecture, we created and integrated unique pre-fetching techniques to reduce the memory access latency and to facilitate real-time processing of our proposed hardware accelerators.

To the best of our knowledge, we could not find any similar work in the published literature that provides FPGA-based hardware accelerators/architectures for CO-based SVM on embedded devices, nor could we find any similar work that proposed system-level architectures, which is imperative for machine learning applications for real-world scenarios.

Our proposed embedded hardware accelerators for CO-based SVM executed up to 79 times faster than their software counterparts on the embedded processor, and also executed up to 35 times faster than the equivalent software running on a desktop computer. This significant performance improvement was due to several hardware optimization techniques incorporated into our embedded hardware architectures, including creating customized and optimized architectures by exploiting inherent parallelism and pipeline nature of the computations/tasks; designing computations/tasks

to overlap with memory access; burst transfer and pre-fetching techniques. Furthermore, our embedded hardware accelerators achieved up to 100% classification accuracy. It was also observed that our embedded hardware accelerators achieved better performance (with a constant number of iterations to find the minima for the convex optimization, requiring much less number of iterations) compared to that of the desktop computer. From the results and analysis of our proposed hardware accelerator, it was also observed that the accuracy results and the speedup results increased with the increasing data size. These performance metrics are crucial especially for real-time machine learning applications on resource-constrained embedded devices.

Considering the resource utilization values (i.e., occupied area on chip), it was observed that those values were compatible with our previously proposed efficient FPGA-based hardware accelerators for applications with similar computation complexity on embedded device [44], [45], [47]. Since pre-fetching techniques integrated to reduce the memory access latency added extra hardware resources, it is important to consider the speed-space tradeoffs, especially in embedded devices with their limited hardware footprint.

These experimental results are encouraging and indeed show a great potential in creating and utilizing FPGA-based hardware architectures to support and accelerate machine learning applications, specifically on embedded platforms. The compact size of our proposed accelerators as well as the ability of our embedded architectures to dynamically train from the unstructured datasets, further enhance the potential of deploying machine learning applications on embedded devices. Currently, we are exploring the most recently proposed optimization techniques for SVM [7] and deep

neural mapping SVM. We are also planning to provide dynamic reconfigurable hardware accelerators [36], [37], [46] for machine learning applications to integrate smart and adaptive traits to our designs.

Power consumption is another major issue in resource-constrained embedded devices. However, in this chapter, we do not report power, mainly because Xilinx Power Analysis tools, for the ML605 platform with Virtex-6 FPGA, reports estimated power, which is not necessarily accurate. Also, FPGAs typically consume less power than embedded processors [10]. Regardless, we are planning to investigate sophisticated power analysis tools to measure power consumption of our FPGA-based hardware accelerators on embedded devices.

In this chapter, we introduced novel, unique and optimized embedded hardware accelerator architecture for data classification using SVM on portable and mobile embedded devices. SVM is one of the popular data classification algorithm used in machine learning. Applications such as medical diagnosis, ionosphere data analysis, SVM can be used to categorize efficiently by reducing the generalization errors. We discussed the implementation details for both embedded software and embedded hardware architecture. We performed experiments to illustrate the feasibility of scalability of the accelerator modules suitable for large-scale real-time data processing.

CHAPTER 4

SYSTOLIC ARRAY ARCHITECTURE FOR SUPPORT VECTOR MACHINES

Many machine learning applications have found their way into portable mobile devices, which have limited resource availability. In this chapter, we investigate parallel processing architecture for machine learning applications, since our previous work [219],[220],[221] on FPGA-based parallel processing architectures for compute/data-intensive applications demonstrated significant performance-gain with the penalty of area. Considering these tradeoffs, we introduce a novel, unique and efficient hardware architecture to accelerate support vector machines using systolic array configuration on embedded platforms. We evaluate the feasibility, efficiency, and scalability of our hardware architecture utilizing both embedded software and hardware optimization techniques. Our design is generic, parameterized and adaptable with various numbers of systolic arrays for machine learning applications. Our proposed hardware design achieves 107 times speedup compared to its software counterpart and can also achieve 100% classification accuracy.

4.1 Introduction

Machine learning is a method of data analysis that enables a system to automatically learn and perform a specific task without explicit instructions. It is part of Artificial Intelligence (AI), for extracting valuable information from large volume of data using algorithms and statistical models [15]. Due to the benefits of identifying important insights in data, machine learning has spread into various fields of applications such as cyber-security, health-care, transportation etc. In addition, the capabilities and flexibility of mobile and embedded devices are steadily increasing to incorporate various machine

learning techniques, which in turn pose serious challenges and design limitations such as stringent area, memory and power limitations.

Machine learning can be broadly classified into two categories: supervised learning (classification) and unsupervised learning (clustering). In supervised learning, pre-labeled datasets are used to train a system to perform specific task whereas in unsupervised learning, the system is trained by clustering the data into different classes without using pre-labeled datasets. However, supervised learning is popular due to the availability of large volume of data. As the volume of data size increases, the computing time to train a system increases. In order to handle such enormous amount of data, the machine learning algorithms are becoming more complex, requiring significant processing power. A typical general-purpose processor is inefficient to cater for such substantial processing power and the existing machine learning algorithms (currently suitable for processor-based design only) are not executable as is on embedded platforms. Consequently, some kind of new hardware architectures is required to support machine learning applications on mobile and embedded devices.

Our main objective is to provide optimized hardware architectures and techniques to accelerate machine learning applications on embedded mobile devices. In this chapter, we focus on Support Vector Machines (SVM), which is one of the most popular classification algorithms in machine learning applications. The hardware accelerator is optimized to accelerate classification methods by providing a dedicated hardware design and customizing the execution overhead compared to general-purpose processor, thus improving area, power and speedup performance. We make the following contributions in this chapter: generic, parameterized and scalable hardware accelerator for support

vector machines, unique optimization techniques including pre-fetching, burst transfer, parallel systolic array configuration, adaptable and efficient system-level design suitable for different fields of machine learning applications.

Support Vector machines is a classification algorithm developed by Cortes and Vapnik in 1995 [28]. It involves training and testing process [24]. During the training phase, the SVM classifier constructs a hyper-plane separating two different classes of datasets. In geometry, a hyper-plane is a subspace (in our case, a 2D-plane) with (n-1)-dimensions compared to n-dimensional ambient space. The margin width of the hyper-plane determines the distance measure of classification between two separate classes. Higher the margin width implies higher accuracy of data classification. Margin width of the hyper-plane can be increased using mathematical optimization techniques [137] such as convex optimization [23]. Furthermore, increasing the margin width of the hyper-plane using convex optimization and efficient techniques to solve convex problems is convenient for SVM classifier to handle large compute-intensive and data-intensive applications; however, a general-purpose process is incapable to cater for high processing power requirements. In addition, the formulation of the SVM utilizes the mathematical kernel techniques [94], which further enable the SVM classifier to classify non-linearly separable data. During the testing phase, the new unclassified data are passed to the trained SVM model utilizing the sign verification step to determine the respective class.

For real-world applications, the characteristics of the datasets vary based on different factors. In order to compensate for different applications, the classifier must be generic to accommodate these variations in the datasets. Therefore, SVM provides 3 most popular mathematical kernels (linear, polynomial, Gaussian kernel) suitable for linearly

separable and non-linearly separable data. Choosing the right mathematical kernels and the specific parameters determined by cross-validation method [24] can help to achieve high accuracy results. Furthermore, the convex optimization formulation is solved by using the sequential minimal optimization approach. This approach chooses two data point at any instance to determine the direction of the gradient descent to find the minima value. Using the minima values, the convex optimization maximizes the margin width on the hyper-plane. Since, this is an iterative process; hardware design can take advantage of parallelism inherent in the SVM algorithms. Also, a hardware accelerator would avoid the execution overhead such as fetching and decoding as in processor-based design, thereby improving the overall area, power and speedup performance. Therefore, it is imperative to develop dedicated hardware architecture to make use of SVM iterative process to accelerate machine learning applications.

4.2 RELATED WORK

Table 1 Literature review

Ref	Contribution	Platform	Resources Utilization	Acc	Speedup	Pow
[85]	Systolic chain of PEs, pre-computed support vectors	FPGA, Xilinx Virtex-5	5162-slices, 64-DSP, 329kB-mem, 74-BRAM	88 %	~33 fps	N/A
[138]	Training phase with reconfig. Arch-DT,SVM,ANN	FPGA, Xilinx Virtex-7	1062-slices, 12-DSPs	N/A	42.79-66.71x	N/A
[43]	Training-Matlab, Decision boundary condition-HW	Matlab, Xilinx xc5vlx110t	2010-33360:slices, 0-515:DSP, 27-63:IOBs	N/A	N/A	1.4-2
[139]	Multiplier-less kernel-systolic array, offline training MATLAB	Matlab	N/A	N/A	N/A	N/A
[48]	Co-processor design, SMO decomposition	FPGA, Xilinx Virtex-5	27735-37549LUTs, 32-128DSP, 16-32BRAM	99.12	18-21x	10 W
Proposed design	Systolic array – kernels, solver, testing	FPGA, Xilinx Virtex-6	8390-slices, 236-DSP, 118-BRAM	95 %	107x	3.42

Recently, hardware acceleration using FPGAs has gained interest due to ease of programmability. We surveyed the existing research work on hardware support for SVM

classifier. In this regard, we investigated ways to utilize FPGAs to design, develop, and implement high-dimension, large-scale data classifier. From the survey papers [140][141][142], we have extracted the related existing research work based on systolic array configuration.

The paper [85] presents first steps towards realization of generic systolic chain of processing elements (PEs) for SVM. The performances of the systolic chain of PEs are evaluated for image and video applications. The paper presents architecture details of distributed pipelined architecture, efficient management of memory and data transfers, fan-out complexity of routing input samples to systolic array modules. Systolic chain of computing elements are designed and implemented for SVM decision boundary condition. We believe extending the work to SVM training would result in considerable improvement in terms of speedup, due to the inherent parallelism of SVM training process.

In [138], an ensemble type of architecture was presented to implement decision trees, neural networks and SVM and evaluated using 18 benchmark datasets. The authors provide a comparative analysis in terms of speedup performance for various machine learning algorithms. The analysis can be used to identify the similar computing procedure used by many machine learning classifier. Identifying the similar computing procedure to perform classification might provide insights to choose the critical computations for systolic array implementation. Thereby, the most time consuming computations can be accelerated by instantiating multiple systolic arrays.

An FPGA-based design was proposed for decision boundary conditions using multiplier-less kernel implementation technique for image processing in [43]. In this

case, only the classification step of the SVM was implemented using the proposed multiplier less techniques to reduce power. However, the training process of SVM was computed offline using MATLAB, which in many cases requires the hardware support to improve the computational time. The support vectors obtained from the MATLAB was stored in the FPGA's internal memory and utilized for decision boundary conditions. In these scenarios, incorporating the multiplier less kernel technique might be useful in order to reduce power but which in turn might add execution time for the overall decision conditions. Extending the multiplier-less kernel approach to the training process on hardware might lead to additional execution time at the cost of overall training performance. The power consumption was reported in the range of 1.479 to 2.051W for the multiplier-less kernel implementation.

In [139], the author present multiplier-less kernel operation for SVM using software based systolic array on Matlab. Using Matlab, the computed α values for support vectors are virtually stored in the systolic array chain for test samples. The processing elements (PEs) mainly consisting of adders and sub tractors are used to reduce the computational complexity of matrix multiplications and presented a reduction from O(n3) to O(n). Based on the presented reduction in computational complexity, it is evident that hardware design for systolic array seems to improve the performance significantly for large-scale datasets.

In [48], a parallel implementation of FPGA as coprocessor was proposed for SVM. With this design, the speedup obtained was around 20x compared to CPU design consuming 10W power. Although, the design utilize PCI and FPGA as a co-processor to

parallelize compute-intensive arithmetic operations, it is not suitable for mobile embedded platforms, which has stringent area and power restrictions.

Based on the aforementioned existing work, implementation techniques are limited to part of the SVM's process. In most cases, the support vectors are pre-computed and used for decision boundary conditions computation using systolic arrays. Therefore, our proposed work in this paper aims to extend the systolic array to the complete SVM training and testing process (including mathematical kernels, convex optimization and boundary decision conditions). Extending these concepts, can help reap the actual benefit inherent to parallel processing capabilities of FPGAs and SVM. In the following section, we provide the system-level details and experimental platforms used to design parallel systolic array for support vector machines.

4.3 Design Approach

In our design, the proposed hardware architecture is implemented using hierarchical and modular-based approaches, in which the SVM algorithm is categorized into 3-level of abstraction. The higher-level involves training and testing process of SVM, followed by mathematical optimization level at the second level. The lower abstraction level includes the fundamental components for arithmetic and vector computations. The aforementioned abstraction structure facilitates resources sharing at different level. The components for arithmetic and vector computations are implemented using the single-precision floating point units to optimize for area and power without compromising the accuracy of the results. These operators are provided by Xilinx IP core libraries.

4.3.1 Experimental Platform

Our proposed SVM accelerator IP is developed using Xilinx's Virtex family FPGAs [103]. Specifically, virtex-6 XC6VLX240T-1FFG1156 FPGA [143][105], which consists of high-performance logic slices (37680 slices) with embedded hardware IP resources such as 748 DSP48E1 slices, 512MB DDR3-SDRAM and soft microprocessor capabilities.

For our experiments, the DSP slices are used efficiently to improve the parallel processing capabilities and logic slices provides a foundation for programmable platform alternative to ASIC design. Soft microprocessor, i.e., Micro-blaze, is used for handling the control signals between customized-IP for SVM, UART serial ports, AXI communication protocols, DDR3-SDRAM, AXI timers and Interrupts [144], [117].

The RTL is design using mixed hardware description language (both VHDL and Verilog) [145] and as proof-of-concept, we have also developed embedded software algorithm in C++. Xilinx Platform Studio (XPS) [146] [147] [148] is used to build the base system-level design and the RTL is designed and evaluated using Xilinx ISE and ISim tools [149] [150] [151]. The embedded software is developed using Xilinx Software Development Kit (SDK), which is based on Eclipse open source standard. Our experimental results are also compared with the baseline python program results (Scikit-learn), to verify the correctness of the results. The overall system-level architecture is illustrated in the fig. 4.1.

4.3.2 Framework of Embedded Hardware

Micro-blaze is a 32-bit soft processor based on RISC architecture optimized by Xilinx for efficient FPGA implementation and available as part of Xilinx embedded

development kit (EDK) [152] [153] [154]. For our experiments, we have selected the area optimization, after synthesis, which translates to 5 DSP48E1 slices and 38 BRAM operating at 100MHz. The micro-blaze initiates the AXI timer [120] and data pre-fetching process using AXI Master Burst transfer [155].

The automated design to access datasets considered for the current experiments [122] are stored in the block ram memory [106]. Initially, the data is passed through mathematical kernel block to distinguish linearly separable and non-linearly separable data sets. Then, the output matrix of the mathematical kernel is passed through the convex optimization solver as shown in the fig. 2. The solver finds the minima value and stores the corresponding α and bias values in the DDR3 using AXI master burst controller and at the same time signaling the micro-blaze to indicate the training process for SVM is completed.

The new data is then passed through the testing block to classify the data into respective classes. Although, the training process for SVM is compute intensive and time consuming, the testing is process is data-intensive depending on volume of the new incoming data. The major step for testing process involves sign verification step and sorting the data into assigned class. Therefore, in order to fulfill both the requirements for compute-intensive training process and data-intensive testing process, we have implemented parallel systolic array configuration for SVM as shown in fig. 2.

Systolic arrays can effectively make use of SVM's massive parallelism of compute-intensive and data-intensive training and testing process. Especially, with efficient implementation on FPGA devices, the systolic array can achieve better speedup performance compared to fixed modules or general-purpose architecture designs.

Our proposed design for systolic array implementation for SVM is favorable for the following 3 reasons.

- First, the high demand for processing power for SVM can be distributed among individual clusters of a systolic array.
- Second, our design is parameterized and scalable thus; the FPGA can be configured to occupy significantly less area for specific applications.
- Third, the framework of shared-resources is the most appealing aspect when considering the mobile embedded devices.

Taking into account of the aforementioned advantages, we performed experiment on our parallel systolic array design by varying the number of instances, execution overhead including single-instruction stream, multiple-data stream (SIMD) and multiple-instruction stream and multiple-data stream (MIMD) and results are presented in the next section.

To evaluate the accuracy and performance of our embedded design, we performed experiments using two benchmark datasets from UCI machine learning repository [121]. The two datasets are as follows: Ionosphere datasets [122] and Wisconsin breast cancer diagnostic dataset [21]. Ionosphere dataset consists of radar data obtained by using 16 high-frequency antennas. The returned complex electromagnetic signal is processed using an autocorrelation function to determine the presence of any structures in the earth's ionosphere. This datasets consists of 351 instances with 34 attributes and suitable for binary classification. Cancer datasets is obtained by determining the features of a cell nucleus of either malignant or benign cancer cells. Cancer dataset represents the characteristics of cell nuclei using 30 different attributes and 569 instances.

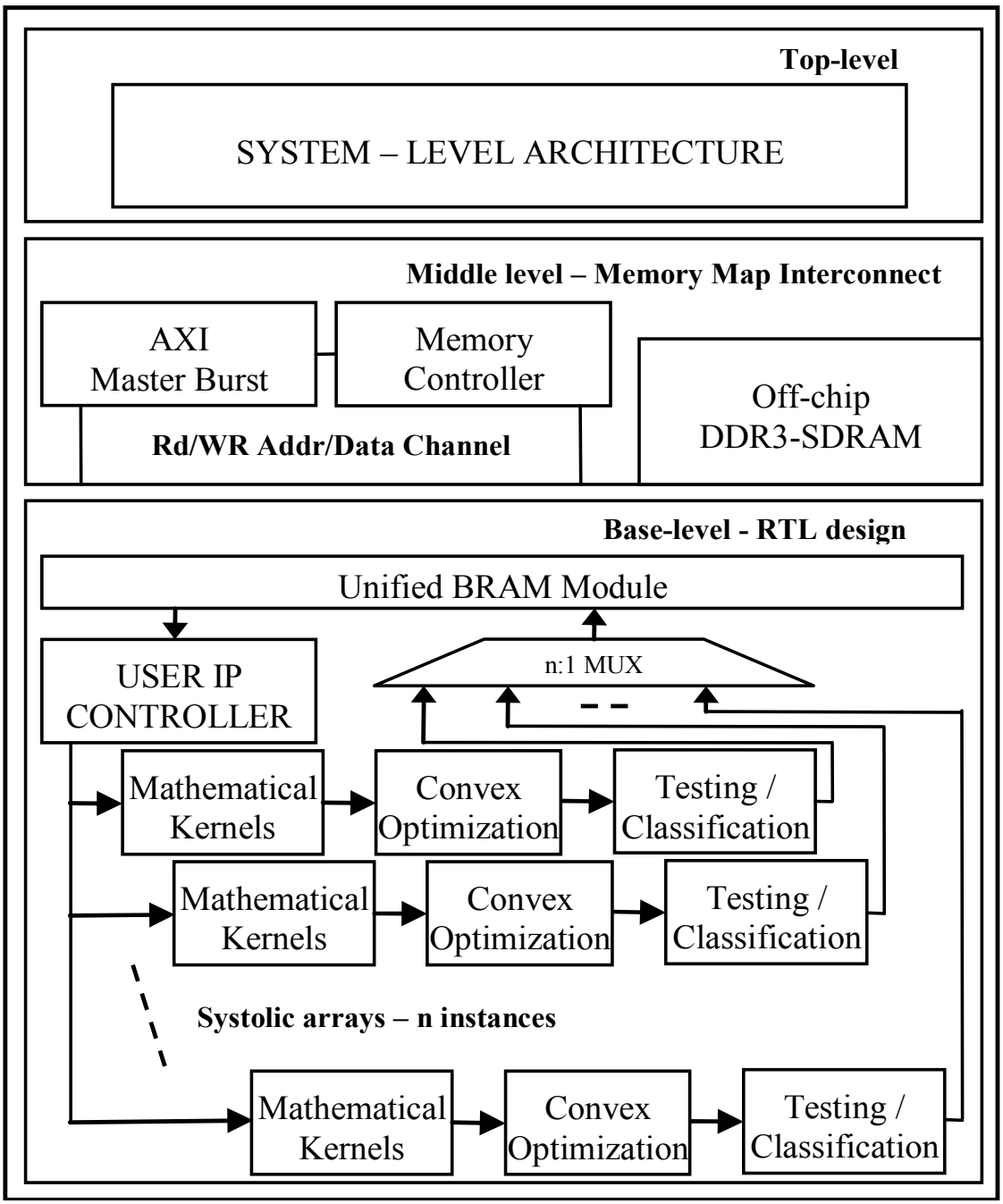


Figure 4.1: Three high-level stages of SVM algorithm

4.4 Experimental Results

We performed experiment to evaluate the speedup performance and feasibility of our proposed design. The speedup performance is measured in real-time using the AXI

timer operating at FPGA's clock frequency at 100MHz. The speedup for each systolic array is calculated using the equation (1) as micro-blaze execution time over embedded hardware execution time.

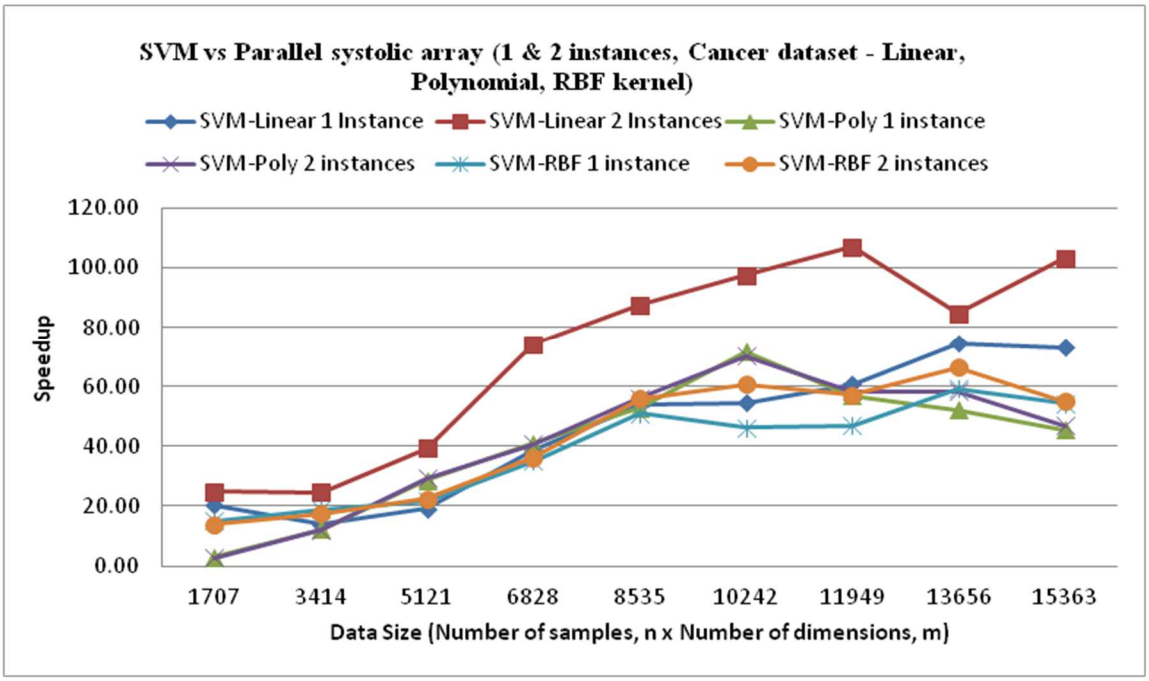


Figure 4.2: Speedup vs. Data size

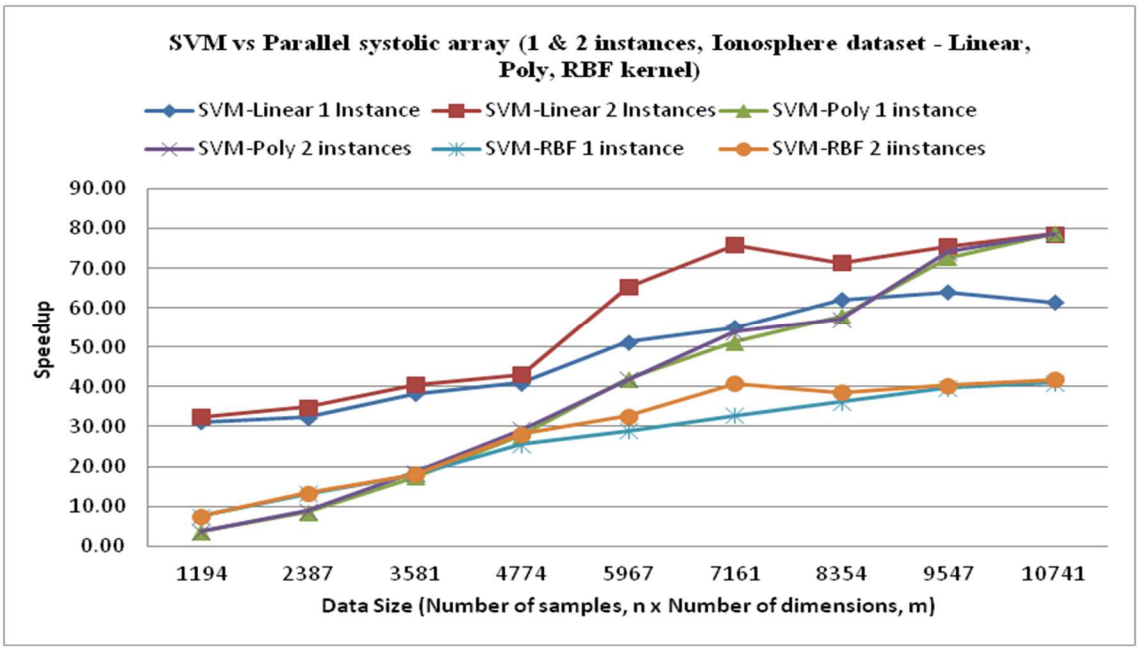


Figure 4.3: Speedup vs. Data size

Table 16. Execution time, speedup and accuracy for Cancer dataset – Linear Mathematical Kernel

Train Set	Micro-blaze Exec Time (s)	HW 1-inst Exec Time (s)	Speedup	HW 2-inst Exec Time (s)	Speedup	Acc (%)
10 %	161865353	7924113	20.43	6526249	24.80	82.76
20 %	325818822	23388548	13.93	13290771	24.51	92.85
30 %	659149652	34461989	19.13	16781132	39.28	91.03
40 %	1341043549	34498506	38.87	18081648	74.17	91.54
50 %	2117706879	39129865	54.12	24178951	87.58	92.26
60 %	3105162265	56985371	54.49	31804238	97.63	91.68
70 %	4296218440	70827842	60.66	40131605	107.05	92.47
80 %	5772976007	77416316	74.57	68077162	84.80	91.32
90 %	7557691210	103336947	73.14	73204946	103.24	92.96

Parameters: C = 2, degree (d) = 2, gamma (γ) = 0.005, Maximum # of iterations = 1000

The accuracy of the SVM classifier is a measure of the total number of correct data classification divided by the total number of test data, as given in the equation (2). The datasets are partitioned evenly to evaluate the accuracy of SVM classifier and the results are compared with the baseline results from open-sources python code [132]. Partitioning the datasets for training and testing purposes provided us a method to interpret the effect of implementing different number of systolic array and the overall efficiency of the convex optimization module.

Table 17. Execution time, speedup and accuracy for Ionosphere dataset – Linear Mathematical Kernel

Train Set	Micro-blaze Exec Time (s)	HW 1-inst Exec Time (s)	Speedup	HW 2-inst Exec Time (s)	Speedup	Acc (%)
10	326247324	10468579	31.16	10026519	32.54	64.51
20	994127844	30749019	32.33	28474615	34.91	63.21
30	2014535135	52676038	38.24	49606272	40.61	66.39
40	3562645809	86721348	41.08	82743009	43.06	59.8
50	5251044535	102352939	51.30	80376897	65.33	70.36
60	7620470915	138372431	55.07	100358129	75.93	84.06
70	10483075447	168745617	62.12	146884860	71.37	98.45
80	13975767881	218558954	63.95	184837167	75.61	98.45
90	19475226254	316911548	61.45	247726125	78.62	96.29

Parameters: C = 2, degree (d) = 2, gamma (γ) = 0.005, Maximum # of iterations = 1000

Table 18. Resource utilization

Number of BRAM	Number of DSP48E1 slices	Number of occupied slices
118	236	8390

Table 1 and 2 presents the embedded software and hardware execution time. The embedded hardware is measured for 2 parallel systolic array instances. We varied the number of instances in the range of 1, 2, 4, 8 and found out that 2 systolic array was sufficient to accelerate the SVM classification for our specific datasets. The table 1 & 2 present the results for Linear kernel, however a brief comparison of speedup for the 3 mathematical kernel (linear, polynomial, RBF) are shown in the figure 4.3 and 4.4. Figure 4.7-4.10 demonstrates the impact of number of instances with respect to hardware execution time and speedup for cancer datasets. The impact of systolic array was relatively low for polynomial kernel as compared to other linear and RBF kernel and it is evident from the figure 4.8-4.10.

$$Speedup = \frac{Software\ Execution\ Time}{Hardware\ Execution\ Time} \quad (25)$$

$$Accuracy\ in\%, (y_i, \hat{y}) = \frac{i}{n_i}\sum_{i=0}^{n_i-1} I(y_i == \hat{y}) * 100 \quad (26)$$

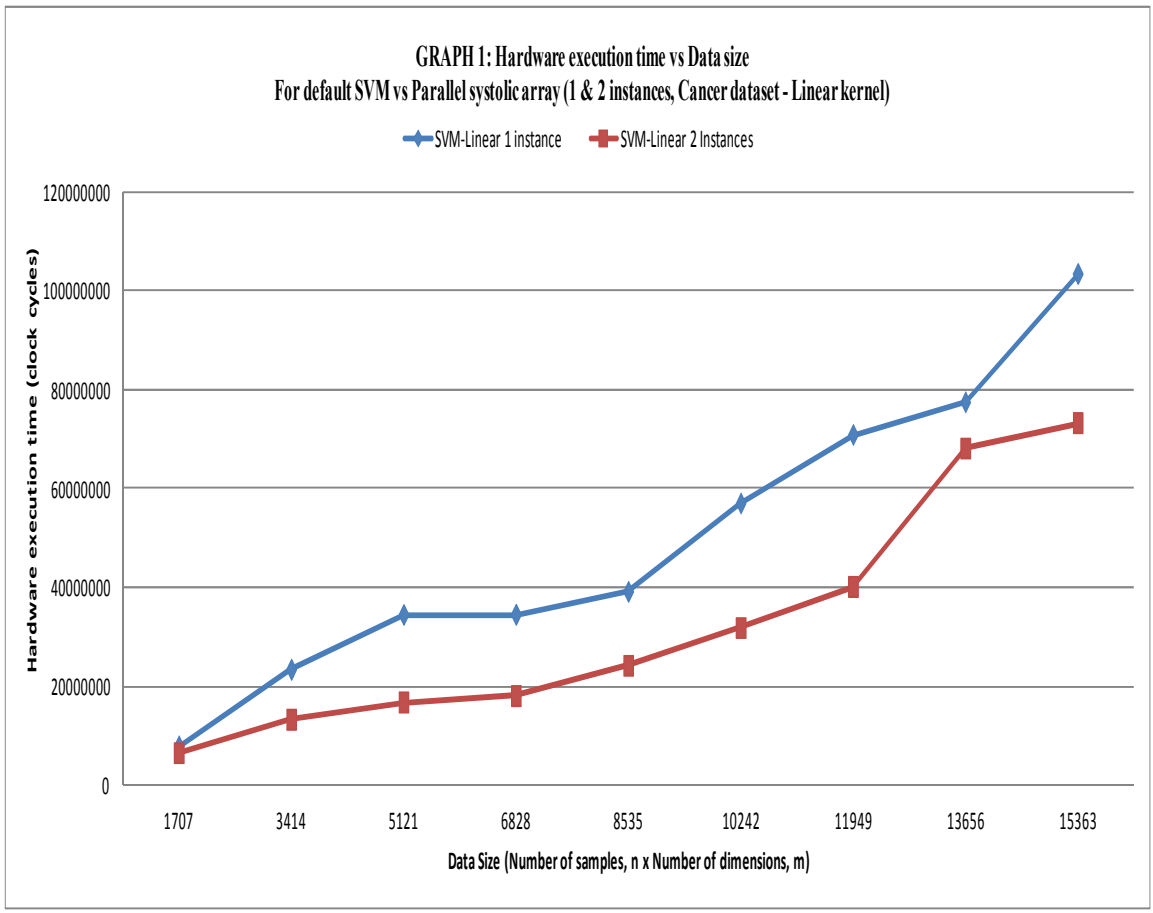


Figure 4.4: Graph of hardware execution time vs. data size for 1 and 2 instance – Cancer datasets, Linear Kernel

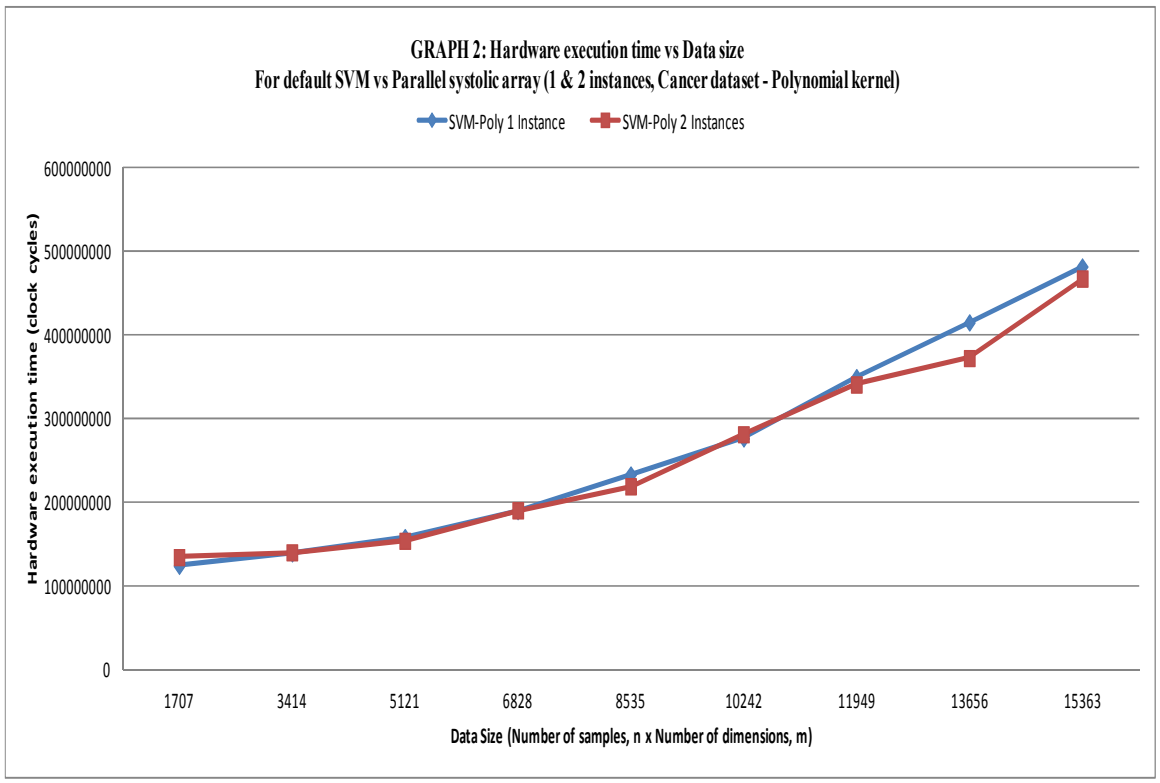


Figure 4.5: Graph of hardware execution time vs. data size for 1 and 2 instance – Cancer datasets, Polynomial Kernel

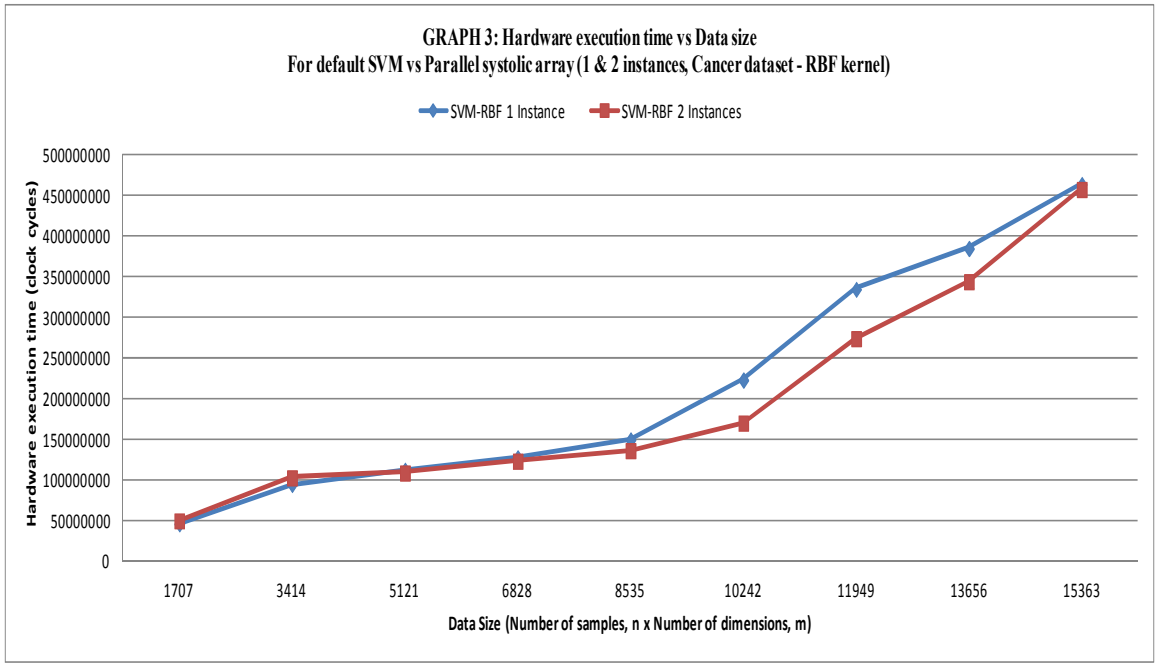


Figure 4.6: Graph of hardware execution time vs. data size for 1 and 2 instance – Cancer datasets, RBF Kernel

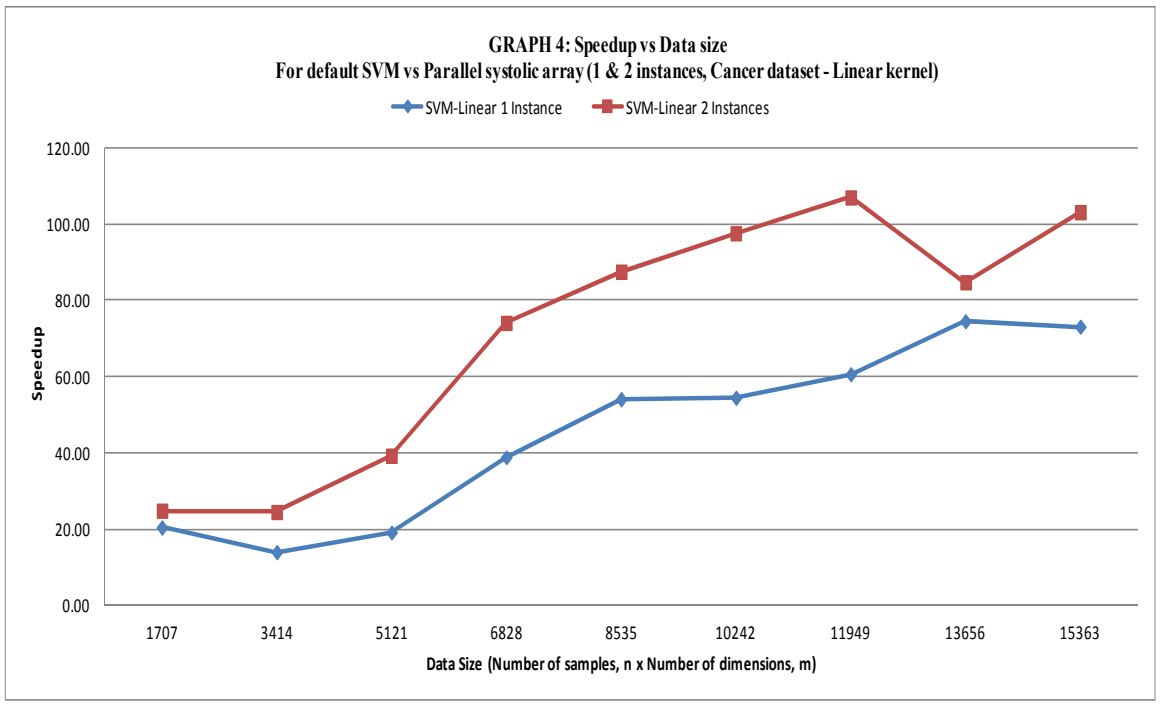


Figure 4.7: Graph of speedup vs. data size for 1 and 2 instance – Cancer datasets, Linear Kernel

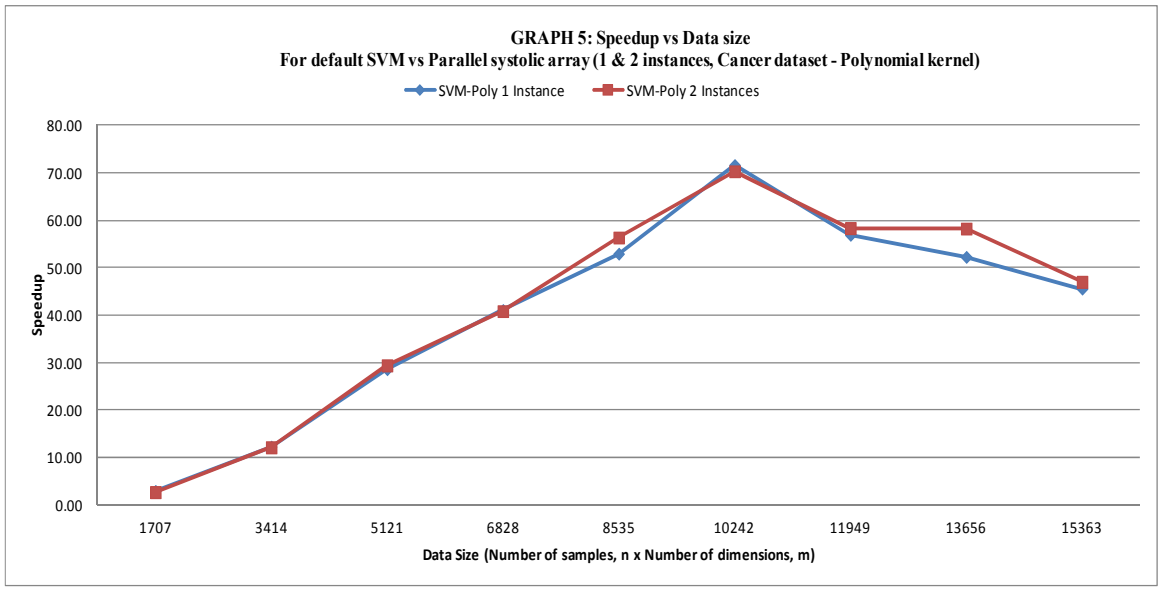


Figure 4.8: Graph of speedup vs. data size for 1 and 2 instance – Cancer datasets, Polynomial Kernel

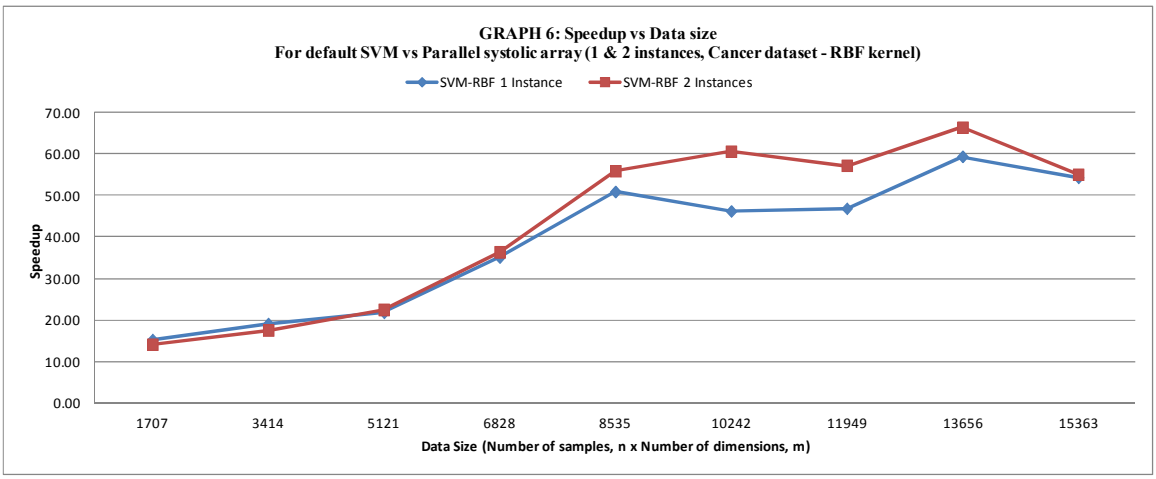


Figure 4.9: Graph of speedup vs. data size for 1 and 2 instance – Cancer datasets, RBF Kernel

Table 19. Execution time and speedup for Cancer dataset – with 2 instances

Hardware execution time (clock cycles) 2 Instances			**Speedup w.r.t Micro-blaze Software**			**Speedup w.r.t 1 instance hardware**		
Lin	**Poly**	**RBF**	**Lin**	**Poly**	**RBF**	**Lin**	**Poly**	**RBF**
6577614	131620218	49506714	24.61	2.75	14.13	1.20	0.94	0.93
13069473	139466677	102632928	24.93	12.19	17.50	1.79	1.00	0.92
16710370	154645119	112085381	39.45	29.26	21.76	2.06	1.02	1.00
20149006	193305613	121217907	66.56	40.23	36.89	1.71	0.98	1.05
25688549	224863725	136752785	82.44	54.95	55.67	1.52	1.04	1.09
31937629	274041069	169782113	97.23	72.18	60.70	1.78	1.01	1.31
48320738	330319529	273059077	88.91	60.20	57.42	1.47	1.06	1.23
68624072	374548485	343348766	84.12	57.91	66.57	1.13	1.11	1.12
76708889	458258150	453057020	98.52	47.95	55.59	1.35	1.05	1.02
Average Speedup			67.42	41.96	42.91	1.56	1.02	1.08

As mentioned, in the previous section, the parameters which determine the orientation of the hyper-plane are set as C=2, degree of the polynomial as 2 and a gamma value of 0.05 with a maximum number of iteration of 1000 for convex optimization are selected. It is observed that as the number of available data for training the SVM model

increases, the accuracy also increases thus increasing the execution time. Based on the systolic array implementation, the execution time is reduced significantly and also number of occupied slices is reduced due to resources sharing facility. Using a single instance systolic array, a speedup of 74 is achieved, however, with the 2-instance systolic array; a speedup of 107 was observed compared to the embedded software architectures.

Table 20. Execution time and speedup for Cancer dataset – with 4 instances

Hardware execution time (clock cycles) 4 Instances			**Speedup w.r.t Micro-blaze Software**			**Speedup w.r.t 1 instance hardware**		
Lin	**Poly**	**RBF**	**Lin**	**Poly**	**RBF**	**Lin**	**Poly**	**RBF**
6621474	136250557	49637011	24.45	2.65	14.09	1.20	0.91	0.93
13301979	142121941	103138928	24.49	11.96	17.41	1.76	0.98	0.92
16711440	157291069	112237912	39.44	28.77	21.73	2.06	1.01	1.00
20531068	198159327	121679926	65.32	39.24	36.75	1.68	0.96	1.05
25645414	226679427	136976330	82.58	54.51	55.57	1.53	1.03	1.09
32312715	281685159	170012310	96.10	70.22	60.61	1.76	0.98	1.31
40686304	342195776	274202777	105.59	58.11	57.18	1.74	1.02	1.22
68652288	371140304	344115110	84.09	58.44	66.43	1.13	1.12	1.12
73692686	462429729	457586208	102.56	47.52	55.04	1.40	1.04	1.01
Average Speedup			69.40	41.27	42.76	1.58	1.00	1.07

Table 21. Advanced Extensible Interface Master Burst Transfer configuration

IP Configurations	**Parameters**
Maximum Burst Length	256
Address Pipeline Depth	14
Master Length Width	20
Native Data Width	32

Table 22. Execution time and speedup for Cancer dataset – with 8 instances

Hardware execution time (clock cycles) 8 Instances			Speedup w.r.t Micro-blaze Software			Speedup w.r.t 1 instance hardware		
Lin	**Poly**	**RBF**	**Lin**	**Poly**	**RBF**	**Lin**	**Poly**	**RBF**
6671329	142512261	49690130	24.26	2.54	14.08	1.19	0.87	0.93
13361585	144867026	103227704	24.38	11.73	17.40	1.75	0.96	0.92
16817152	163123213	112488198	39.20	27.74	21.68	2.05	0.97	1.00
20623317	200390070	122103503	65.03	38.80	36.63	1.67	0.95	1.04
25779519	228932812	137639892	82.15	53.97	55.31	1.52	1.02	1.09
32478974	286973507	170022001	95.61	68.93	60.61	1.75	0.96	1.31
41046446	343419034	275560187	104.67	57.91	56.90	1.73	1.02	1.21
69248797	376423788	345679309	83.37	57.62	66.13	1.12	1.10	1.11
74076625	484891351	457607716	102.03	45.32	55.04	1.40	0.99	1.01
Average Speedup			68.96	40.51	42.64	1.57	0.98	1.07

4.5 Chapter Summary

In this chapter, we introduced efficient and unique embedded hardware architecture with parallel systolic array configuration to accelerate the convex optimization-based support vector machines for data classification in machine learning. Our embedded hardware architectures are adaptable, generic and scalable. Hence, without modifying the internal architecture, our embedded designs can be utilized for different field of machine learning applications involving data classification. We discussed the implementation details for different number of systolic arrays for hardware architecture. We performed experiments to illustrate the feasibility of scalability of the accelerator modules suitable for large-scale real-time data processing.

CHAPTER 5

OPTIMIZED HARDWARE ARCHITECTURE FOR DEEP NEURAL NETWORKS

Deep learning is a subset of Artificial Intelligence (AI), which aims to build smart systems capable of performing tasks that typically requires human intelligence. The smart systems utilize statistical and mathematical approaches to analyze data to perform a task without human interruption. Due to the accumulation of large volume of data over the past decade, the time required for deep learning applications to perform a task has increased exponentially. In order to improve the execution time for deep learning applications, various software optimizations and hardware support has been proposed. The scope of this research work is to examine the feasibility of a programmable device to aid the deep neural network acceleration. Our previous work on machine learning applications has demonstrated significant improvements in terms of speedup performance for support vector machines. In this research work, we will focus on hardware support for deep neural networks using programmable logic devices.

Hardware platforms such as ASIC, CPU, GPU and FPGAs have been integrated to build a neural network, among which GPUs and FPGAs play major role for acceleration. However, despite the high cost and excessive power consumption, GPUs dominate the acceleration for neural networks. Our objective seeks to provide a customized FPGA-based architecture to support deep neural networks for power-efficient design. In this research, we aim to illustrate the design methodologies to leverage the flexibility & power efficiency of a programmable device, present RTL-level details and system-level architecture and lastly examine the potential gains in terms of energy and

speedup performance. Experiments are performed using benchmark datasets for medical diagnosis and radar data analysis.

5.1 Introduction and Motivation

Recent breakthroughs in the development of Neural Networks (NNs) have led to state-of-the-art performance for various deep learning applications [156][157][158]. Neural Networks have been among the most powerful and widely used techniques in cognitive applications. DNNs have outperformed classical techniques and are found in a wide range of applications such as Aerospace & defense, financial services, healthcare, retail, IT & telecom sector etc [159][160][161][162]. However, Deep Neural Network-based designs are complex and computationally intensive. The complex topologies of DNNs with many layers and numerous of parameters have escalated the cost of power consumption. In order to deploy these technologies in mobile devices, area and power are the major constraints and pose a serious challenge to integrate into portable embedded devices.

As the advancement in silicon technology is reaching its theoretical limits, any improvements on typical architectures is unable to keep pace with the computational requirements of deep learning. Hardware acceleration plays a crucial role for real-time operations. In 2010, Microsoft research group [3] proposed augmenting CPUs with configurable platforms to enable deep learning applications. Eventually in 2016, Microsoft incorporated these configurable platforms only for deep learning inference. Despite the high cost, fixed architecture and high power consumption, GPUs dominate the acceleration for training a deep learning model.

As large scale applications (such as scientific computing, social media and financial analysis) gain prominence, the computations and storage demands of modern systems have far exceeded the available resources. It is expected that, in the next decade, the amount of information managed by world-wide data centers will grow 50-fold, while the number of processors will increase only by 10-fold. In fact, the electricity consumption of just the US data centers have increased from 60 billion kWh in 2016 to 73 billion kWh in 2020 [163]. It is clear that, a raising performance demands will soon outpace the growth in resource budgets; hence, over provisioning the resources alone will not solve the conundrum that awaits the computing industry in the near future.

In this research work, we introduce energy-efficient FPGA-based hardware architecture for DNN acceleration with on-chip high speed data transfer technique, scalable IP modules to handle large-scale data and a generic design suitable for various applications. We present the systematical approach to explore tradeoff analysis for area and power cost by varying the number of parallel processing elements and minimizing the memory access latency.

This chapter is organized as follows: In sub-section 5.2, we present the necessary details of deep neural network algorithm and existing work related to FPGA accelerators. In sub-section 5.3, we present the comparative analysis of various platforms and evaluation plan for steps considered to develop accelerators. Our experimental results and analysis are reported and discussed in sub-section 5.4. In the last sub-section, we summarize our work and present concluding remarks for the chapter.

Our objective is to provide customized & optimized hardware architectures to support and accelerate deep neural network on embedded platforms considering the

associated constraints of embedded platforms. We provide the necessary chip-level details and efficient data accessing techniques to support deep learning applications on portable embedded devices.

5.2 Background Study

Neural networks are continuously evolving [164][165][166] due to various factors; among which the two main reasons for such a rapid advancement of neural networks are due to the availability of large-volume of annotated data and the improvements in the silicon technology to support immense computing power requirements. In this section, we present the background study for deep neural network and explore the existing work to accelerate DNNs using FPGA. In the subsequent section, we present a comparative analysis among different available platforms and tradeoff-advantages relative to specific computing platform.

5.2.1 Deep Neural Networks

Deep Neural Networks (DNNs) are typically a set of stacked layer of networks. In general, DNN consists of input layers, L-hidden layers and an output layer. Each layer consists of n-neuron with weights, bias and sample feature as input. Input layers are parameterized to receive high dimensional data and feeds forward to consecutive hidden layers. The parameters of individual neuron, weights and bias values, are evaluated using mathematical optimizations such as Stochastic Gradient Descent (SGD). SGD are formulated to reduce error function by adjusting the weights and bias values. The errors are back-propagated to the first layer and iterated multiple times until a minimum value is reached for error function. This iterative process of adjusting the weight and bias values is called as training. Once a meaningful set of parameters are obtained, the inputs are

passed through the output layer to make useful prediction and the process is called as inference.

5.2.2 Literature Review

We surveyed the existing research work on hardware support for the DNN classifier. In this regard, we investigated ways to utilize Field Programmable Gated Arrays (FPGAs) to design, develop, and implement high-dimension, large-scale data classifier. From the survey papers [167][168][169], we have extracted the following closely related existing research work.

In [170], the authors present 3 stage pipelined Deep Learning Accelerator Unit (DLAU). The three stage pipelined architecture consists of tiled matrix multiplication unit, part sum accumulation unit and activation function acceleration unit. The purpose of DLAU is to scale up accelerator architecture for large-scale deep learning networks using FPGAs as the hardware prototype as well as to maintain low power cost. The presented results achieves a speedup of 36.1x compared to Intel Core2 processor and a power consumption of 234mW. DNN are trained using Matlab and the corresponding parameters are used for inference which is implemented on Zynq Zed board FPGA. Although, extending the FPGA implementation to DNN training could further improve the speed performance. It should be noted that further detailed comparison for speedup relative to network and tile size would be highly recommended. With power analysis, the paper illustrates the power consumption for individual pipeline stages and memory access module with a total consumption of 234mW.

The paper [171] presents one of the critical and important steps for translating a design into hardware modules. The paper highlights the inefficiency of existing tools

such as HLS, OpenCL and suggests a new RTL complier to efficiently allocate the resources for maximum throughput and performance. Manual RTL translation is time consuming and requires hardware skills. To address the inefficiency gap between high-level synthesis tools and the manual RTL translation, the authors present a scalable solution to achieve near optimal RTL implementation. The proposed compiler is referred to as ALAMO, is optimized based on various design strategy to automatically set modular and scalable modules to accelerate the operations of deep learning. Although, a generic modular implementation for an efficient data transfer would be suitable for continuously evolving DNN architecture but the current performance of HLS or OpenCL tools must be scrutinized for in-depth analysis.

In the paper [172], the authors proposed an alternative methodology, based on scalable hardware architecture and circuit design using stochastic based computing principle and stream-mode compute, to efficiently implement CNN on FPGA that outperforms GPUs in terms of power consumption and performance. The designs were evaluated using Virtex-6 and power consumption is reported at 3.61W. One of the concerns that rises is that when a stochastic based approach is used, extracting and identifying the parameters that has majority contribution for a given network would be useful, i.e., a quantifiable factor represented using an equation.

The paper [173] proposed a FPGA-based accelerator for DNN using Xilinx Zynq-7020 FPGA and evaluated using MNIST digital identification dataset to achieve up 96% recognition rate. It should be noted that the proposed design is limited to DNN inference. No details or implementation are specified for DNN training but the weights parameters are stored in ROM in advance for inference. The memory configuration has to be

addressed in the paper. Some of the implementation seems trivial in terms of DNN topology. Since, most of DNN has extremely complex structural topology and the proposed implementation lacks the scalability aspect of an FPGA design.

In [174], FPGA-based accelerator is presented for CNN inference. The proposed design is based on INT8-2 compute approach using an adaptive logic module to accelerate low-precision inference. It would be of interest in regards to training to further monitor variation in accuracy. Since the approach seems to provide a potential area optimization but an in-depth analysis would be interesting for tradeoff analysis.

Table 23: Literature review for neural networks

Ref	Contributions	Platforms	Resource utilization	Acc. (%)	Speedup	Pow (W)
Our work	SOC Ensemble architecture IPs, On-chip training and inference	Virtex-6	7500-10870-slices, 107-143-DSP, 118-BRAM	91	61x	3.6-4.9
[40]	Multi-layer-perceptron and radial basis ANN. Training phase with reconfigurable arch. for DT, SVM, ANN	Virtex-7	1062-slices, 12-DSPs	NA	66.71x	NA
[170]	3-stage pipeline accelerator unit, tiled mat-mul, training using MATLAB, Inference using FPGAs	Intel processor, Zynq Zed	220-DSPs, 280-BRAM, 53200-LUTs	NA	36.1x	1.814
[171]	Addresses the inefficiency of high-level synthesize tools, RTL complier to allocate max throughput & perf, near RTL implementation	Altera Stratix-V GXA7, OpenCL	256-DSPs, 2330-BRAM, 121k-Logic	80	1.9x	19.5
[172]	Stochastic based computing principle & stream-mode compute	Virtex-6	Freq-200MHz	87	899.13 GOPS	3.61
[173]	Inference using FPGAs, pre-determined weight vectors in ROM	Zynq 7020	12-DSPs,6-BRAMs, 38899-Luts	96	NA	NA
[174]	Low-precision inference using FPGAs, INT8-2 using adaptive logic	Arria 10, Stratix 10	69%-DSP, 30%-ALM, 19%-BRAM	71	200-GOPs	GPU-300

[175]	One single-computing layer for fully-connected computation fabric, 16-bit FP, hard-coded network weights, on-board inference only	Virtex-5 XC5VLX, ZynQ-7000	900-DSPs, 545-BRAMs, 218600-LUTs	98	15.9k FPS	NA
[176]	FPGA accelerator optimized for throughput, Scalable framework with four-level parallelism	Intel i7, Virtex-7 VX690T	3600-DSPs, 1470-BRAMs, 433200-Luts	99	14.84	88-CPU, 225-GPU, 25.6-FPGA
[177]	Latency-driven, weights reloading, SDF transformation, HLS	ZynQ XC7Z045	900-DSP, 545-BRAM	98	1.49x	NA
[178]	Dynamic programming, multi-CLP accel., pipeline, C++ design entry	Virtex-7 VX485T, VX690T	3600-DSP, 2940-BRAM, 84%-LUT	NA	2.3x	11.2
[179]	2D systolic array automation flow for on-board inference only, 8-16 FP	Arria 10 GX 1150	1518-DSP, 2713-BRAM, 83%-LUT	98	NA	20.75
[180]	Numerically characterizing loop opti. Technique, tradeoff analysis relative to memory latency	Intel Arria 10	100%-DSP,70%-BRAM,38-LUT	NA	5.5x	30.44
[181]	Scalable and flexible parallel PEs, quantization, on-chip buffers	ZynQ XC8Z020, XC7Z045	220-DSPs, 140-BRAMs, 56%-LUTs	NA	6x	24.1
[182]	Co-Processor design, parallel 2D primitives, off-the-shelf PCI implementation	Virtex-5 LX330T	192-DSP, 324-BRAM	NA	31x	0.61
[183]	Int. factorization, CNN complier translates high N/W specs to parallel microprogram.	Virtex-5 SX240T	1056-DSPs, 516-BRAM, 48-bit fixed point	-	6.5x	1.14
[184]	Custom processing tiles, fast stream memory interface	Virtex-6 VLX240T	100% DSP, 416-Mem	-	133.6x	14.7
[185]	NoC SW config., mutli-bank memory modules, 32-bit FP C design entry	Stratix-V GSM D5	1590-DSP, 2014-BRAM	-	3x	7.15

Based on the above mentioned existing work, we aim to introduce novel, customized, and optimized FPGA-based hardware architecture for deep neural networks on embedded platforms. Our architecture aims to address the major constraints associated

with the embedded platforms. The area, power consumptions, design reconfiguration, time-to-market are the major constraints associated with the embedded platforms. We also introduce lean and compact embedded software architecture for DNN, which is designed to fit into the available resources of the embedded microprocessor on chip. The experimental results are encouraging and indeed show a great potential in utilizing FPGA-based systems to support and accelerate deep learning applications, specifically on embedded platforms. The compact size of our proposed architectures as well as the ability of our embedded designs to dynamically train from the unstructured datasets, further enhance the potential of implementing deep learning on embedded devices.

5.3 Design Approach and Development Platform

In this section, we present the details of available platforms to accelerate DNN, provide an insight on the current technology trend and the steps to develop acceleration architecture of a programmable device.

5.3.1 Comparison of Different Platforms

In this section, we discuss various means of computing platforms suitable for different application domains. For instance, most of the computations of neural networks involve floating point operations and matrix computation. The execution flow and performance comparison on two popular platforms GPU, FPGA are as follows:

- GPU: GPUs are designed to accelerate certain operations such as multiple-and-accumulate (MAC) operations. The architecture is based on SIMD (single instruction multiple data) architecture to support large number of MAC, matrix operations etc. Recently, Nvidia started to incorporate additional module called, Tensor core, to accelerate specifically for neural networks computations. GPUs

offer immense computing power to handle data-intensive and compute-intensive applications. However, they are power hungry devices. For any specific applications, the power consumption of GPUs are almost 30x-50x more compared to CPUs or FPGAs. Although, GPUs can perform a lot faster than CPU or FPGA, the architecture is fixed and the power consumption is too high.

- FPGAs: FPGAs are one of the promising avenues for exploring tradeoff analysis for area and power. It is a suitable platform to efficiently prototype a design. Due to their inherent nature of re-programmability, a number of different applications can be efficiently developed and prototyped to meet faster time-to-market requirements. FPGAs also offer useful solutions to develop complex design such as providing embedded processor, DSP blocks, IP cores, custom IP cores, I/O interfaces etc [186]. The important factors such as low-power requirements, less system cost, adaptable platforms provides an opportunities to design a high-performance computing platforms for large-scale data applications such as deep learning. However, the run-time to translate and map an RTL design takes longer and in certain cases routing the individual blocks fails due to limited interconnect resources. In such cases, the user needs to carefully modify the design to meet the synthesis rules.

Table 24: Comparison of various computing platforms

	CPU	**GPU**	**ASIC**	**FPGA**
Speed	Slow	Fast	Fastest	Medium/fast
Power	Med	Highest	Lowest	Low
Cost	Fair	High	High	Low
Perf/W	Low	Med	High	High
Architecture	Traditional	Compute Pow	Fixed Arch.	Reconfigurable

To get a clear insight, we limit our focus to GPUs and FPGA comparison. An in-depth comparison between GPUs and FPGAs must be considered in order to make a reasonable choice in terms of speed, power consumption, and cost. Based on the analysis and survey of existing work [187][188][189][190][191][192][193][194][195][196], we present a speculative analysis to get an insight for GPU and FPGA comparison as illustrated in fig 5.2. The corresponding table 25 summarizes the contributing factor for each partition/range. In the graph, X-axis is the size of an applications, Y-axis is theoretical estimate for performance based on multiple factors such ease-of-use, power, cost, performance. The graph is partitioned into three segments based on the cost of the GPU/FPGA ranging from:

- Partition 1: Low-end range, less than 10k
- Partition 2: Mid-range, between 10k and100k
- Partition 3: Data centers, higher than 100k USD.

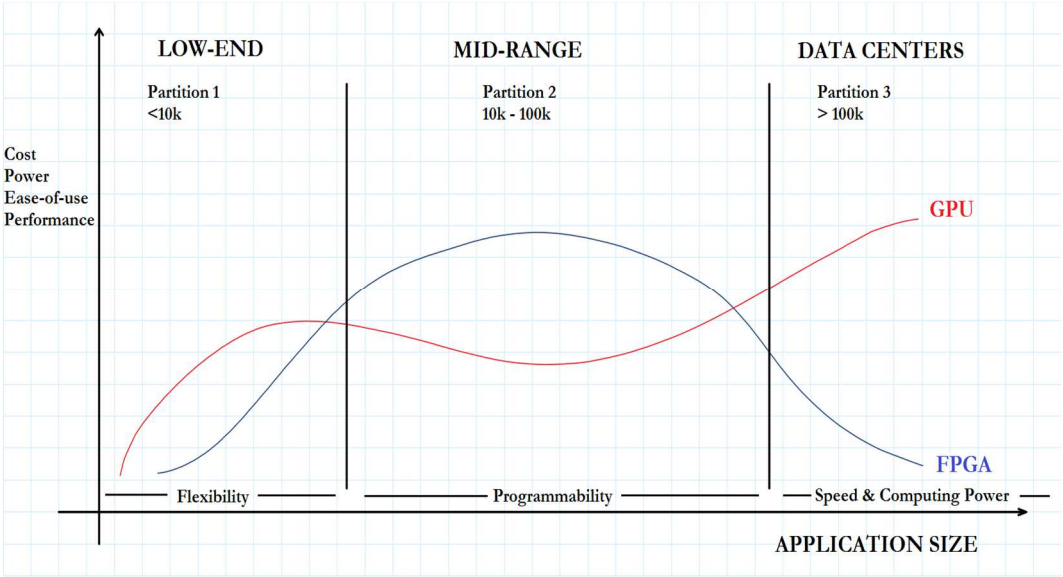


Figure 5.1: Speculative analysis for GPU vs. FPGA comparison relative to application size

Partition 1: In this range, choice of GPU is more suitable compared to FPGAs, because the main contributing factor is the ease-of-use. Not much modification is

required to run on the GPUs which are less time-consuming, no hardware expertise is required and the perf/W gain is negligible. Almost same performance can be obtained from FPGAs but requires time-consuming hardware design skills.

Partition 2: This is the range where the choice of GPUs and FPGAs becomes a tradeoff. For researcher or third party vendor with their own servers, they can extract the potential gain from FPGAs in terms of performance per Watt. With certain hardware design skills, a designer can make the most in terms of cost and flexibility to adapt to various applications. For instance, we can directly connect a camera module to FPGAs and make a real-time face recognition system. On the other hand, we need a host computer to use GPUs for real-time face recognition. Therefore, FPGAs provide a lot more flexibility when compared to other platforms. In case, if the designer plans to fabricate their own chips, FPGAs provides that flexibility as well to prototype the chip prior to fabrication.

Table 25: Tradeoff analysis for GPU & FPGA

LOW-END	MID-RANGE	DATACENTERS
- Ease-of-use - Contributing factor - Hardware skills required for FPGAs - Computing power relative to platform price	- Cost & Perf/W - Contributing factor - Reconfigurability to incorporate various optimization techniques - For example, Approx. computing approach for FP operations - Towards prototyping for ASICs, ASIPs - Needs to take care of additional overhead for area, power and cost - Choice of GPUs and FPGAs will be tradeoff - Long-term application and application size, budget	- Computing power - Contributing factor - Run-time, High-end FPGAs - NA - FPGAs mostly fails to configure due to rout ability issues and hard to scale up for the large applications - AI Training - GPUs - AI Inference - FPGAs (recently being incorporated) - Approximate cost analysis using data centers service charge calculator - GPU - ~39 cents per million images (scaled value) - FPGAs - ~21 cents per million images (scaled value)

Partition 3: Anything over $100k, these are high-end models for data centers. Currently, not many FPGAs can support the applications at the Datacenter's range. Also, in the cloud computing platforms such as Microsoft Azure, Amazon AWS we can find FPGAs only for the AI inference but not for AI training. Because translating the design, routing the design using limited interconnects in the FPGA will most typically fail at that level. Therefore, to the best of our knowledge, we could not find FPGAs that can handle the computing power required by the cloud computing platforms.

5.3.2 Design Approach and Evaluation Plan

Many other platforms can be parameterized and configured to perform many tasks. However, when it comes to designing a dedicated hardware or a chip, we need flexibility to decide on architectural trade-offs to meet the design budget. Flexibility relative to the logic elements, number of clock cycles, number of transistors etc (in most cases, the design budget is very stringent). Therefore, to turn an algorithm into a chip, using above mentioned parameters we can identify the main contributing factors to improve the performance and cut back on some of the parameters to meet the design budget. Therefore, the following experimental results illustrate an analysis for potential optimized acceleration solutions.

- In the conventional CPU implementations, frequent delays arise due to the communication between CPU & memory. In some cases, using hardware accelerations can decrease performance due to, cost of moving the data is larger than the gain from the faster hardware execution. These problems can be easily reduced in FPGA device by providing large amount of local memory space.

- Earlier implementations rarely used multiplication in FPGA since it was fairly expensive in terms of resources. Newer FPGA families targeted for many digital signal processing applications include high-speed hardware multipliers and multiply accumulate units. Making use of all the major advantages of FPGAs over CPU devices, we can apply this design to many application domains.
- On the other hand, FPGA comparison with ASIC involve addressing the continuously evolving changes in the neural networks. In the last decade, the neural networks have undergo a number of changes and optimization and keeping pace with such rapid changes in terms of hardware design will add significant cost to deploy a design on hardware.

In summary, in order to support and handle the computation requirements of neural networks, GPUs and FPGAs are the potential solutions. The trend in the hardware technology is following towards ensemble architecture with both GPUs and FPGAs. For the time being, FPGA seems like a reasonable choice to adapt for a wide range of applications at low cost, power consumption.

5.3.3 Base System-level Design and Internal Architecture for Deep Neural Network

An L-layer DNN can be represented mathematical [197] as shown in equation (1) below:

$$F(x;\theta) = (f_L \circ f_{L-1} \circ \dots \circ f_1)(x) \quad (1)$$

where, x: input sample

θ: weight parameters, w_i

F: output function

The overall output function in equation (1), F can be simplified as shown in equation (2),

$$F = w_{i+1} \circ f_i \circ \alpha_{i-1} \tag{2}$$

As mentioned previously, an error function is formulated to adjust the weight parameters using stochastic gradient decent. The mathematical equation of error function for classification is given in equation (3).

$$J_C(x, y; \theta) = -\langle y, (\log \circ \sigma)(F(x; \theta)) \rangle \tag{3}$$

The error function is minimized by iterating through a loop by means of back-propagation. The stochastic gradient descent approach is used to find the minimum error function values using derivatives, as represented in the equation (4) below:

$$\nabla_{\theta_L} J(x, y; \theta) = \nabla^*_{\theta_L} f_L(x_L) . D^* w_{L+1}(x_{L+1}) . e_L = \nabla^*_{\theta_L} f_L(x_L) . e_L \tag{4}$$

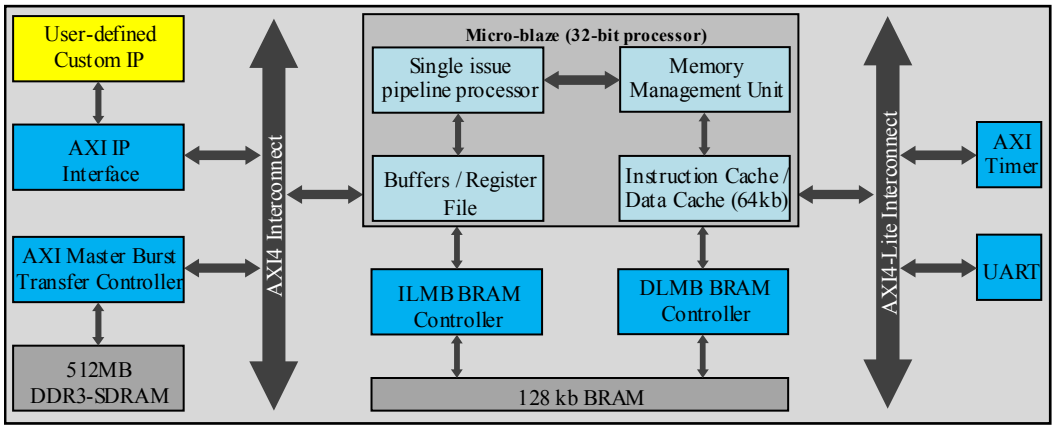


Figure 5.2: Base system-level architecture for deep neural network

The system-level of higher abstraction level is shown in figure 5.3. The base system-level design is almost same as mentioned in chapter 3, except the internal architecture of user-defined custom IP. The user-defined custom is IP is shown in figure 5.7. The above generic mathematical framework for L-layered DNN is calculated using derivatives with respect to the parameters of each layer to minimize error function. DNN has extremely high structural complexity; however, the underlying fundamental operations involve matrix-vector computations as illustrated in figure 5.4 & 5.6.

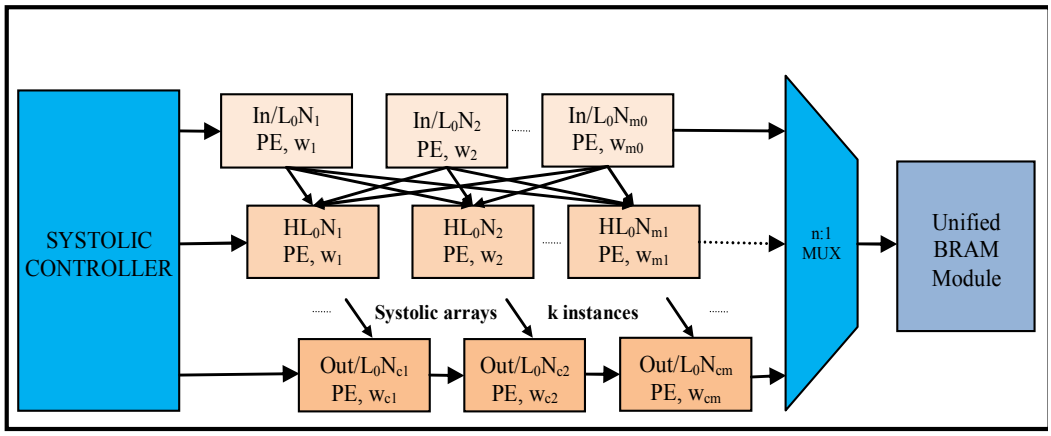


Figure 5.3: Internal Architecture and Data-flow for k-layer deep neural network

The dataflow in a DNN, results in a long dependencies due to chain of back-propagation but the matrix computations and convolution operations remains unchanged. Matrix computations and convolution operations contribute to total of 90% of the overall operations [167].

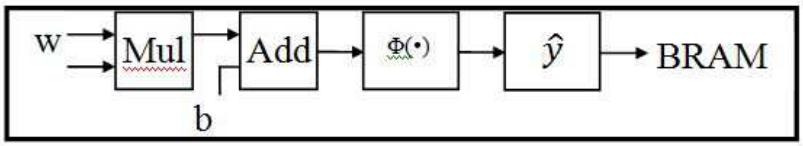


Figure 5.4: Internal Architecture of single neuron

Constructing a highly parallel computing array to support the common matrix computations would efficiently accelerate the training process, as illustrated using a single generic module in Figure 5.5. Therefore, in this research our objective seeks to provide customized and optimized configurable hardware architectures to support both inference as well as training process at low-cost for portable embedded devices.

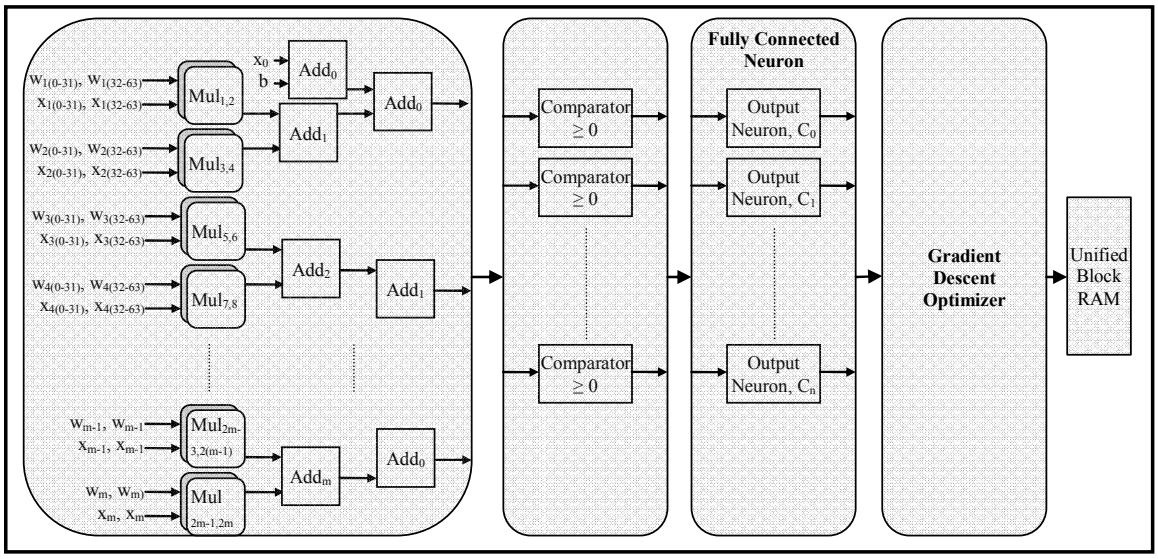


Figure 5.5: Internal Architecture for Input Layer of Deep Neural Network

Unlike GPUs, we estimate that developing a generic, parameterized and configurable architecture has considerable merit to solve the scalabilities issues for continuously evolving computational demands of a deep neural network.

5.3.4 Experimental Platforms

The accelerator IP for the neural networks is designed using mixed hardware description language (both VHDL and Verilog) and as proof-of-concept, we have developed DNN model using software codes (Spyder software). Xilinx Platform Studio (XPS) is used to build the base system and the Register-transfer Level (RTL) DNN will be designed using Xilinx ISE (Integrated Synthesis Environment) & ISim tools. Our experimental results are also compared with the baseline software results, to verify the correct operation.

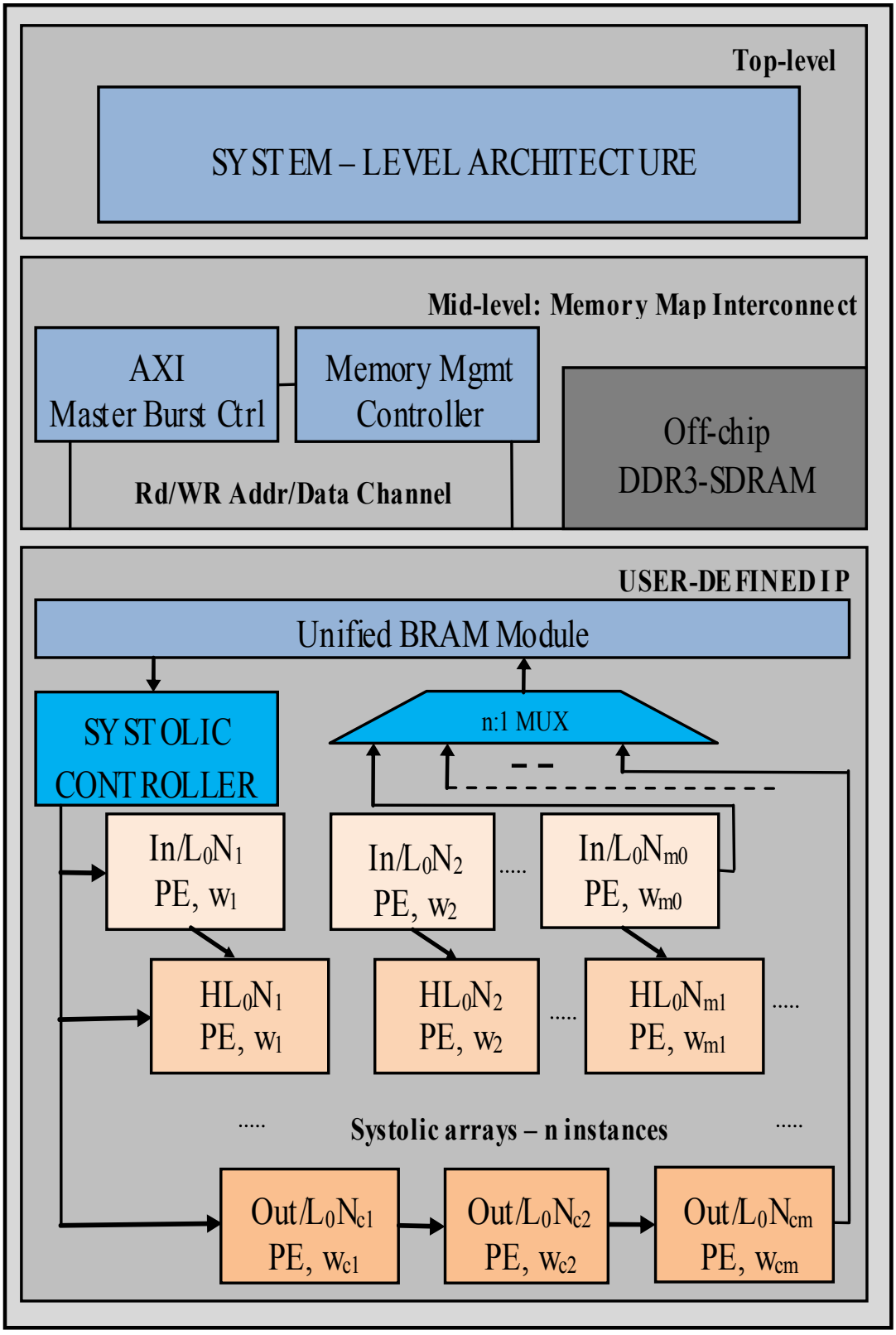


Figure 5.6: High-level multiple layers deep neural network

The proposed DNN accelerator IP is developed using Xilinx's Virtex 6 family FPGA [103]. Specifically, XC6VLX240T-1FFG1156 FPGA [198], which consists of high-performance logic slices (37680 slices) with embedded hardware IP resources such as 748 DSP48E1 slices, 512MB DDR3-SDRAM and soft microprocessor capabilities. For our experiments, we utilize DSP slices efficiently to improve the processing capabilities and the logic slices provides a foundation for programmable platform for the RTL design. Soft microprocessor, i.e., Micro-blaze, is used for handling the control signals between customized-IP for DNN, UART serial ports, AXI communication protocols, DDR3-SDRAM, AXI timers and Interrupts.

MicroBlaze is a 32-bit soft processor based on RISC architecture, available in Xilinx embedded development kit (EDK). For our experiments, we have selected the area optimization for MicroBlaze. Synthesis of MicroBlaze translates to 5 DSP48E1 slices and 38 BRAM operating at 100MHz. The micro-blaze handles the initialization tasks for pre-fetching using AXI Master Burst transfer [199] and AXI timer [120]. The automated design to access data [122] from DDR3-SDRAM is stored in the BRAM [200].

5.4 Experimental Results and Analysis

In this section, we present the results and analysis for cancer diagnosis datasets. The accuracy of the DNN is a measure of number of correct data classification over the total number of test samples, as given in the equation (5). The datasets are partitioned in a range of 10 - 90% for training and rest of the data is used to evaluate the accuracy performance of DNN classifier. Partitioning the datasets for training and inference provides us a method to interpret the tradeoff analysis and the overall efficiency of acceleration modules.

$$Accuracy\ in\ \%, (y_i, \hat{y}) = \frac{i}{n_i} \sum_{i=0}^{n_i - 1} I(y_i == \hat{y}) * 100 \qquad (5)$$

5.4.1 Case 1: Analysis on Classification Accuracy with Limited Iterations

Table 26 presents the comparison analysis between our work on SVM and the neural network accelerator for relative comparison. In this experiment, we have divided the benchmark dataset into varying ratio of training and inference samples. The SVM and NN are trained with 10 - 90% of the dataset with an increment of 10% and the rest of the data used for inference to evaluate the accuracy performance.

Table 26: Accuracy & Exec time for cancer dataset classification using SVM & NN with limited iterations

Training set (%)	MicroBlaze Exec time SVM (sec)	HW Exec Time SVM (sec)	MicroBlaze Exec time CNN (sec)	HW Exec Time CNN (sec)	CNN Accuracy (%)
	RBF	RBF			
10	0.04391	0.0102643	0.036265	0.007827879	38.86
20	0.086489	0.0128378	0.0684011	0.007932034	66.6
30	0.208964	0.01655639	0.3263731	0.019670391	78.9
40	0.24173	0.021692	0.6072149	0.021347582	84.37
50	0.82645	0.0281505	1.27713	0.03476139	86.523
60	1.09582	0.0301093	1.74163	0.034667513	87.89
70	1.28426	0.0315973	1.880129	0.032668579	88.67
80	1.96756	0.0325584	2.286097	0.037346288	90.42
90	2.68965	0.03374492	2.8942913	0.049193527	91.01

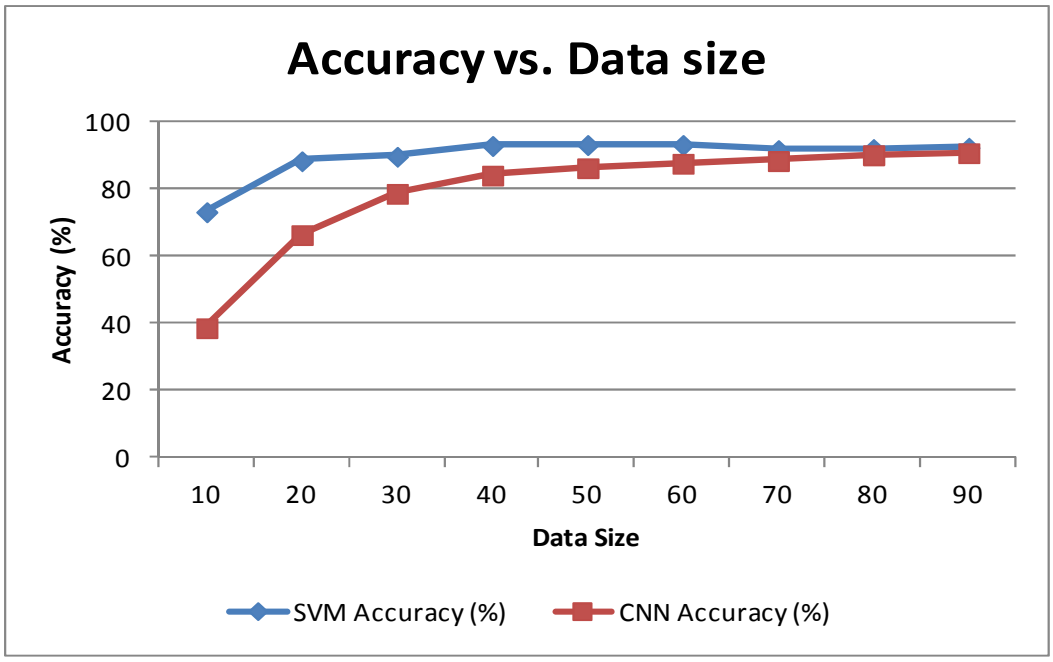


Figure 5.7: Plot of accuracy vs. training data size of cancer dataset using SVM and NN with limited iterations

Table 27: Accuracy & Exec time for cancer dataset classification using SVM & NN with Unsynchronized trails

Training set (%)	MicroBlaze Exec time SVM (sec)	HW Exec Time SVM (sec)	MicroBlaze Exec time CNN (sec)	HW Exec Time CNN (sec)	CNN Accuracy (%)	SVM Accuracy (%)
	RBF	RBF				RBF - Emb
10	0.04391	0.0102643	10.045617	0.217640163	98.2143	73.45
20	0.086489	0.0128378	10.839301	0.224211084	97.3451	88.63
30	0.208964	0.01655639	11.3788097	0.209691984	98.2353	89.96
40	0.24173	0.021692	14.1634256	0.18567331	99.1189	93.19
50	0.82645	0.0281505	20.820615	0.235712636	97.8873	93.47
60	1.09582	0.0301093	26.859118	0.277693702	97.9472	93.48
70	1.28426	0.0315973	28.777553	0.257613382	98.2412	91.96
80	1.96756	0.0325584	29.966104	0.215443676	98.022	92.27
90	2.68965	0.0337449	32.0186489	0.220408166	98.6328	92.66

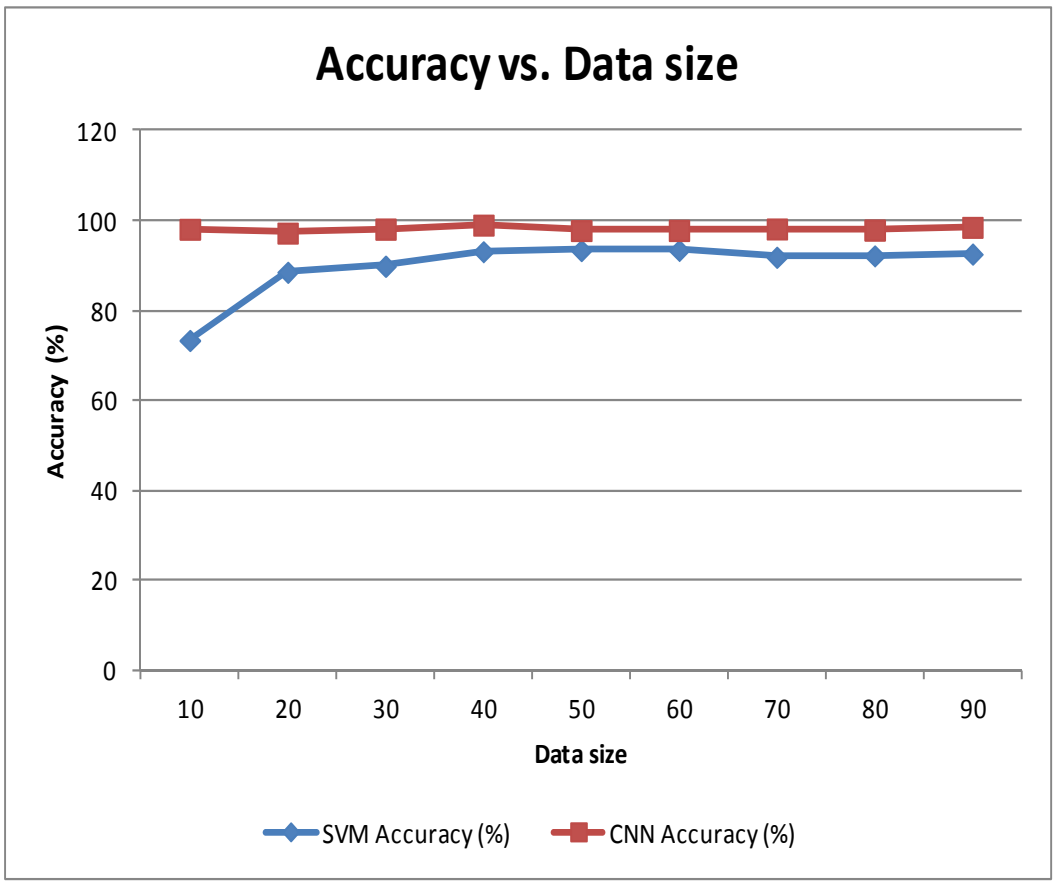


Figure 5.8: Plot of accuracy vs. training data size of cancer dataset using SVM and NN with unsynchronized iterations

5.4.2 Case 2: Analysis on Classification Accuracy with Unsynchronized Trails

For evaluating the execution time and accuracy performance of each classifier is tested for almost the same iteration count as SVM to present a insight relative to difference performance. The accuracy performance of both the classifier is almost same for similar iteration count. However, the execution time required is higher for neural networks compared to SVM as shown in Table 26. These results are further utilized to validate the correct operation of the hardware IPs as illustrated accuracy vs. data size plot (Figure 5.7 & Figure 5.8).

5.4.3 Analysis on Execution Time relative to size of training vectors

The execution time is another major criteria for performance analysis for our proposed embedded hardware accelerators for deep leaning applications. In order to evaluate our embedded hardware accelerators/architectures, we create embedded software architectures for the deep neural network algorithm on the same development platform. Our embedded software architectures are executed on the 32-bit MicroBlaze processor. The execution times for both the embedded hardware and software architectures are obtained using the AXI Timer using cascade timer implementation. These execution times are measured in real-time (sec), while our proposed embedded architectures are actually running (in real-time) on the chip. In this case, we design the AXI Timer in cascade mode to measure the accurate execution times for complete training process. This is mainly because in certain scenarios, especially for large datasets, the execution times exceed the allowable timer counter value of the AXI Timer depending on the number of layers and epochs. In order to resolve the counter overflow issue, the AXI timer is designed utilizing two timers in cascade mode.

The execution times for our proposed embedded architectures for the deep neural networks algorithm are obtained with the varying data sizes. These execution times are presented in Table 28, 29, for the Cancer benchmark dataset, respectively. Similar to the classification accuracy results, three sets of execution times for embedded architectures are obtained separately, for general processor time, micro-blaze execution time and hardware design time. The execution times for each set (for both the embedded hardware and software) are measured 10 times and the average is presented in the aforementioned tables, respectively.

Table 28: Speedup Comparison – Cancer Dataset

Data size	Vect (%)	# Vect	Base-Line	MB Exec time	HW Exec Time (sec)	Acc. (%)	Speedup /BL	Speedup /MB
1707	10	57	0.931	0.8844	0.1132	38.86	0.95	7.81
3414	20	114	0.986	0.9071	0.1014	66.6	0.92	8.94
5121	30	171	0.99	0.9009	0.0540	78.9	0.91	16.67
6828	40	228	1.223	1.0028	0.0518	84.37	0.82	19.36
8535	50	285	1.155	0.8431	0.0259	86.52	0.73	32.54
10242	60	342	1.113	0.7902	0.0189	87.89	0.71	41.65
11949	70	399	1.143	0.7772	0.0162	88.67	0.68	47.85
13656	80	456	1.193	0.7873	0.0140	90.42	0.66	56.19
15363	90	513	2.24	1.3664	0.0221	91.01	0.61	61.74

The execution times for the embedded software and embedded hardware architectures for the deep neural networks are illustrated in Figure 10Figure 5.9, respectively, for the Cancer benchmark datasets. As illustrated, the execution times increase almost exponentially with the increasing data sizes, for both the embedded hardware and software architectures. Somewhat similar behaviors are observed when using the Ionosphere dataset.

For our embedded hardware accelerator/architecture with polynomial kernel, the speedup increases linearly (from 8 times to 61 times faster than the software counterparts) when the percentage of training set increases from 10% to 90%.

The speedups, resulting from the embedded hardware architectures over embedded software, for the deep networks, are presented Table 2828. Figure 5.11 demonstrate the speedups versus the data sizes (percentage of training set) for our embedded hardware accelerators for the deep neural networks for the Cancer benchmark dataset, respectively. At a glance, as shown in Figure 5.11, the speedups typically increase as the percentage of training set increases.

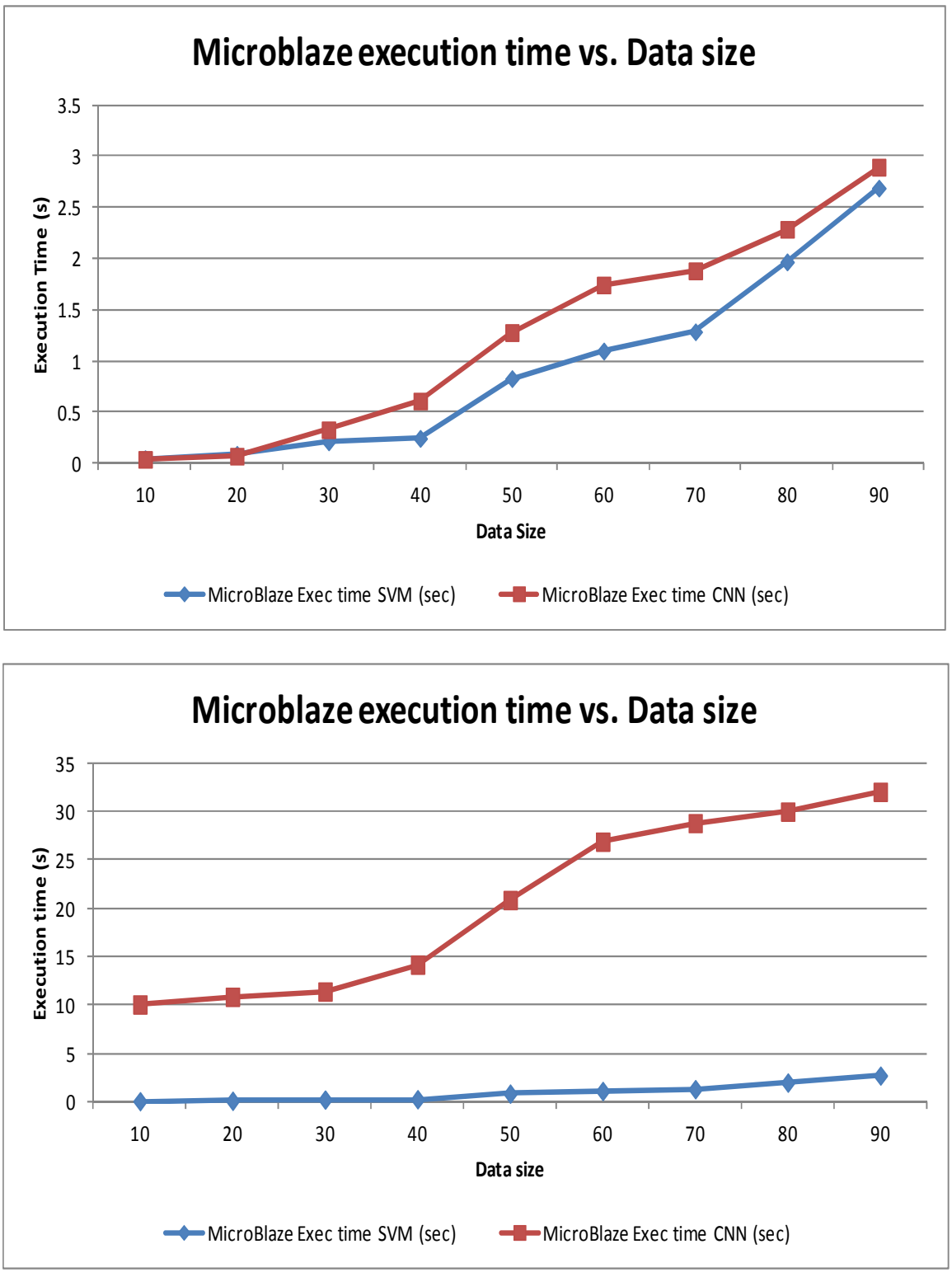


Figure 5.9: Embedded Software: Execution Times vs. Data Size for Cancer Benchmark Dataset, case 1 (above) & 2 (below)

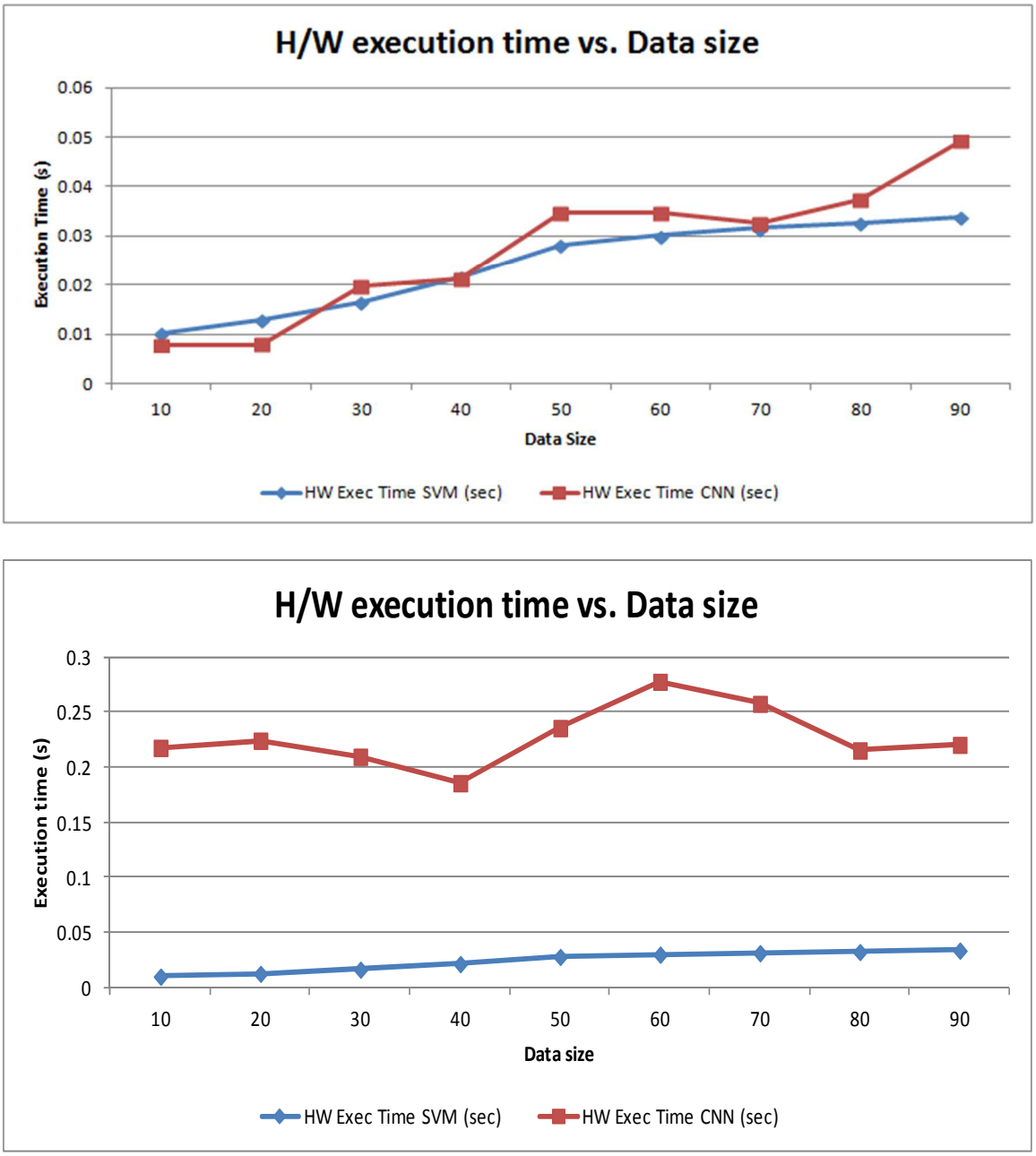
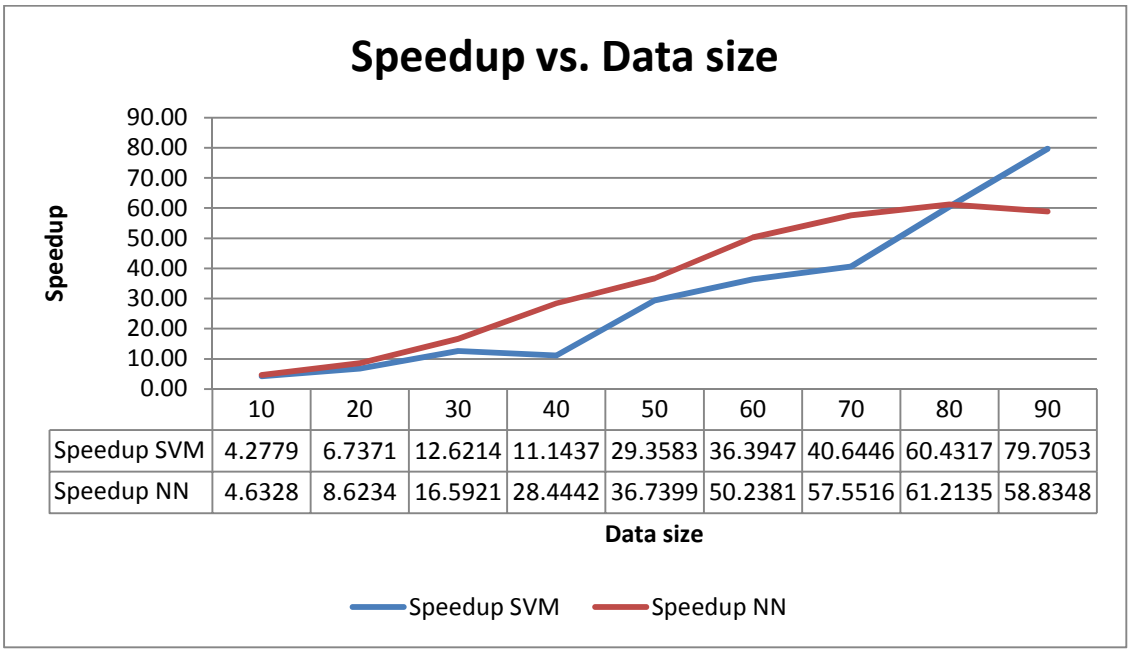


Figure 5.10: Embedded Hardware: Execution Times vs. Data Size for Cancer Benchmark Dataset, case 1 (above) & 2 (below)

As mentioned in the beginning of section 3.4, additional software experiments are performed on a desktop computer using a python code for deep neural network algorithm. Hence, we also compare our embedded hardware accelerators/architectures running at

100MHz on Virtex-6 FPGA with the baseline python software design on the Intel i7 processor running at 2.3GHz. In this case, the results for our proposed embedded hardware accelerators/architectures are shown in Table 27-29 compared to the software design.



	10	20	30	40	50	60	70	80	90
Speedup SVM	4.2779	6.7371	12.6214	11.1437	29.3583	36.3947	40.6446	60.4317	79.7053
Speedup NN	4.6328	8.6234	16.5921	28.4442	36.7399	50.2381	57.5516	61.2135	58.8348

Figure 5.11: Embedded Hardware: Speedup vs. Data Size for Cancer Benchmark Dataset

From these results, it is evident that our proposed embedded hardware accelerators achieve superior speedups (up to 61 times), compared to the equivalent software running on embedded processor on the same development platform. This significant performance improvement is due to several hardware optimization techniques incorporated into our embedded hardware architectures, including creating customized and optimized architectures by exploiting inherent parallelism and pipeline nature of the computations/tasks; designing computations/tasks to overlap with memory access; burst transfer and pre-fetching techniques.

From the aforementioned results and analysis, it is observed that for our proposed embedded hardware accelerators/architectures for the deep neural networks, as the number of samples (i.e., vectors) increases, the accuracy results and the speedup results also increase. In this case, when the classifier has more samples to learn, it could lead to identifying complex patterns. Furthermore, as the size of the matrices is increasing, as well as the complexity of the computations/operations is increasing, customized and optimized embedded hardware architectures might be the best avenue to accelerate and enhance various performance metrics of the CO-based SVM algorithms, compared to the conventional computing platforms such as general-purpose processors.

5.5 Concluding Remarks

In summary, in order to achieve better performance to handle high-dimensional data, software optimization alone cannot provide the required support. It is essential to provide hardware support for these applications with customized embedded architecture. Our architecture aims to address the major constraints associated with the embedded platforms. The area, power consumptions, design reconfiguration, time-to-market are the major constraints associated with the embedded platforms. The experimental results are encouraging and indeed show a potential gain in utilizing FPGA-based systems to support and accelerate deep learning applications, specifically on embedded platforms. The compact size of our proposed architectures to dynamically train from the unstructured datasets, further enhance the potential of implementing deep learning on embedded devices.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In our research work, our objective is to provide new architectures, techniques and design methodologies to overcome the constraints and the requirements associated with big data applications on an embedded platform. We are specifically interested in optimizing the Deep Neural Networks (DNN) and Support Vector Machines (SVM) classifier, using a Convexity approach, which has not been implemented in previous published literature. Our initial investigations reveled that; this capability can be further enhanced using the convex optimization method irrespective of the number of inequality constraints; efficient ways to accelerate the data computing capabilities. As mentioned in section 2, there are several research works on hardware support for general SVM algorithm in the published literature. Most of these hardware architectures were not generic or parameterized. Also, most of the architectures were not designed with portable embedded devices in pretext. None of these works proposed system-level architectures and associated techniques to facilitate the real-time machine learning applications. From our extensive investigation and to the best of our knowledge, we did not find any hardware support or any FPGA-based hardware accelerators, especially for the convex optimization-based SVM algorithm, in the published literature, which makes our research novel and unique.

In this research work, we also introduced novel, customized, and optimized FPGA-based hardware accelerator for deep neural networks on embedded platforms. Our embedded architectures are generic, parameterized, and scalable. Thus, without changing

the internal hardware architectures, our embedded designs can be used for different datasets with varying sizes, can be executed on different embedded platforms, and can be used for various machine learning applications, while satisfying the associated constrains of the embedded devices. We also introduced efficient system-level architecture for our FPGA-based hardware accelerator IPs. With this system-level design, we designed and integrated unique techniques to reduce the memory access latency and to facilitate real-time data analysis and processing.

Our embedded hardware accelerator for CO-based SVM executed up to 75 times (base level architecture) and 107 times (systolic array architecture) faster than its software counterpart running on the embedded microprocessors. Our embedded hardware accelerator for DNN executed up to 61 times faster than its software counterpart running on the embedded microprocessors. Our embedded designs (both hardware and software) achieved up to 100% classification accuracy. These performance metrics are crucial especially for real-time machine learning applications, which typically require processing a large volume of data.

We also introduced lean and compact embedded software architecture for SVM and DNN, which was designed to fit into the available resources of the embedded microprocessor on chip. The experimental results are encouraging and indeed show a great potential in utilizing FPGA-based systems to support and accelerate machine learning applications, specifically on embedded platforms. The compact size of our proposed architectures as well as the ability of our embedded designs to dynamically train from the unstructured datasets, further enhance the potential of implementing machine learning on embedded devices.

6.2 Future work

In this research, we have highlighted many opportunities and obstacles to incorporate in the FPGA-based design. Several large-scale software packages have been developed for many application domains. But some of these domains can be optimized further by appending the above mentioned methods. In future, the existing systems already uses several management schemes such as low-precision computation, pre-fetching, and dynamic scaling to ensure synergy of many techniques and it is also important for a smooth integration of the new techniques in commercial systems. As the quest for performance confronts resources constraints, major breakthroughs in computing efficiencies are expected from many conventional or unconventional approaches.

In this proposal, we have provided the details of system-level architecture for supervised learning, investigated accelerations techniques relative to systolic arrays, and developed an efficient memory management technique. Using the knowledge gained to further establish a framework for instruction set architecture for AI chip design would be valuable contribution to accelerate AI applications and support global digital transformation. However, there are several challenges which still need to be addressed. Some of the potential future directions presented below are worth investigating further.

- One of the challenges we came across to solve quadratic optimization was the selection of hyper-parameters for efficient convergence. Prior to training process, we hope that providing another independent module for model selection criterion based on certain probability estimation could improve the training time as well as reduce the generalization error efficiently.

- In this work, we have developed the embedded architecture for sequential minimal optimization, which restricts to two working sets. For our future work, we intend to develop generalized decomposition methods to solve the sub-problem of quadratic programming.
- Our embedded architecture is developed based on the duality theory formulation. Further investigation is recommended to provide hardware support for primal minimization using interior point method to address the computational burden of iterative process of convex optimization.
- In this research, we have provided the hardware support for convex optimization. We intend to explore semi-definite programming independently for different mathematical optimizations. Also, providing the flexibility to incorporate the hyperbolic tangent kernels.
- Also, as future work, we are planning investigate ways to incorporate partial and dynamic reconfiguration features (as stated in [222],[223]) and HDL code optimization techniques (as stated in [224]) into our FPGA-based accelerators/architectures to further enhance the area-efficiency and flexibility, similar to our partial and dynamic reconfiguration works in [204],[212],[225],[226],[227],[228],[229]. In addition, we are investigating ways to integrate our multi-ported memory architectures, including [230],[231],[232],[233], to facilitate the parallel processing modules in our proposed systolic array designs, to further enhance the speedup, while considering the speed-space tradeoffs.

REFERENCES

[1] Markets and Markets INC, "Embedded System Market by Hardware (MPU, MCU, Application-specific Integrated Circuits, DSP, FPGA, and Memories), Software (Middleware, Operating Systems), System Size, Functionality, Application, Region - Global Forecast to 2025," Markets and Markets, Northbrook IL, 2020.

[2] P. W. Ankita Bhutani, "Embedded Software Market Size, By Operating System (General Purpose Operating System (GPOS){Windows, Linux[Ubuntu, RedHat, Debian, OpenSUSE, Fedora], Android}, Real Time Operating System (RTOS){VxWorks, QNX, FreeRTOS, ARM Mbed}), By Function (Standalone S," Global Market Insights, 2019.

[3] Microsoft Corporation, "Project Catapult," Microsoft, [Online]. Available: https://www.microsoft.com/en-us/research/project/project-catapult/.

[4] O. Elgaw, A. M. Mutawa and A. Ahmad, "Energy-Efficient Embedded Inference of SVMs on FPGA," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Miami, FL, USA,, 2019.

[5] D. Singh and C. Reddy, "A survey on platforms for big data analytics," in *Journal of Big Data 2, 8 (2015). https://doi.org/10.1186/s40537-014-0008-6*.

[6] T. Chen, X. Gao and G. Chen, "The features, hardware, and architectures of data center networks: A survey,," *Journal of Parallel and Distributed Computing,,* pp. 45-74, 2016.

[7] D. Hennessy and J. Patterson, Computer Organization and Design MIPS Edition, Elsevier, 2013.

[8] S. Abdallah, A. Chehab, I. H. Elhajj and A. Kayssi, "Stochastic hardware architectures: A survey," in *2012 International Conference on Energy Aware Computing, Morphou, 2012, pp. 1-6.*.

[9] Clive Max Maxfield, Application-Specific Integrated Circuits (ASICs), ISBN 9781856175074, https://doi.org/10.1016/B978-1-85617-507-4.00017-6., 2009.

[10] S. Mittal and J. S. Vetter, "A Survey of CPU-GPU Heterogeneous Computing Techniques.," in *ACM Comput. Surv. 47, 4, Article 69, 35 pages. DOI:https://doi.org/10.1145/2788396*, July 2015.

[11] S. Bre and M. Heimel, "GPU-Accelerated Database Systems: Survey and Open Challenges," in *Springer Berlin Heidelberg*, Berilin, 2014.

[12] S. Karthikeyan & B. Gandhare, "Survey on FPGA Architecture and Recent Applications," in *2019 International Conference on Vision Towards Emerging Trends in Communication and Networking, Vellore, India, 2019, pp. 1-4.*

[13] Clive Max Maxfield, FPGA vs. ASIC Designs ISBN 9780750689748,, https://doi.org/10.1016/B978-0-7506-8974-8.00004-1., 2008.

[14] Xilinx "Large FPGA Methodology Guide," Xilinx, Inc., San Jose, CA, 2012.

[15] C. M. Bishop, Pattern recognition and machine learning, Springer, 2006.

[16] BCC Market Research - Machine Learning: Global Markets to 2022, "BCC Research," May 2018. [Online]. Available: https://www.bccresearch.com/market-research/information-technology/machine-learning-global-markets.html.

[17] M. Mohsin and D. G. Perera, "An FPGA-Based Hardware Accelerator for K-Nearest Neighbor Classification for Machine Learning on Mobile Devices," in *Proc. of IEEE/ACM International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, (HEART'18)*, 2018.

[18] A. Baheti and D. Toshniwal, "Trend Analysis of Time Series Data Using Data Mining Techniques," in *IEEE International Congress on Big Data (BigData Congress)*, 2014.

[19] P. Berkhin, "Survey of Clustering Data Mining Techniques," Technical Report, Accrue Software, 2002.

[20] E. Alpaydin and C. Kaynak, "Optical Recognition of Handwritten Digits Data Set," UCI Machine Learning Repository. [Online].

[21] W. H. Wolberg, W. N. Street and O. L. Mangasarian, "Breast Cancer Wisconsin (Diagnostic) Data Set - Machine Learning Repository," University of California, Irvine, School of Information and Computer Sciences, 1995. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

[22] C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," ’98.

[23] S. P. Boyd and L. Vandenberghe, Convex optimization, Cambridge, UK; New York: Cambridge University Press, 2004.

[24] C. Campbell and Y. Ying, Learning with support vector machines, vol. 10., San Rafael, Calif.: Morgan and Claypool Publishers, 2010.

[25] N. Deng, Y. Tian and C. Zhang, Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions, Chapman & Hall, CRC, 2012.

[26] L. Wang, Support vector machines: theory and applications, New York; Berlin: Springer, 2005.

[27] V. Piccialli and M. Sciandrone, "Nonlinear optimization and support vector machines," *Springer Nature 2018,* vol. 16, pp. 111-149, 2018.

[28] V. N. Vapnik, The nature of statistical learning theory, 2nd ed., New York: Springer, 2000.

[29] D. Anguita, A. Boni and S. Ridella, "A digital architecture for support vector machines: theory, algorithm, and FPGA implementation," *IEEE Transactions on Neural Networks,* vol. 14, no. 5, pp. 993-1009, Sep. 2003.

[30] S. Venkateshan, A. Patel and K. Varghese, "Hybrid Working Set Algorithm for SVM Learning With a Kernel Coprocessor on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 23, no. 10, pp. 2221-2232, Oct. 2015.

[31] M. Papadonikolakis and C. Bouganis, "Novel Cascade FPGA Accelerator for Support Vector Machines Classification," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 23, no. 7, pp. 1040-1052, July 2012.

[32] "Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? J. Mach. Learn. Res. 15, 1 (January 2014), 3133–3181.".

[33] S. Afifi, H. GholamHosseini and S. and Poopak, "Hardware implementations of SVM on FPGA: A state-of-the-art review of current practice," *International Journal of Innovative Science, Engineering & Technology, (IJISET),* vol. 2, no. 11, pp. 732-752, Nov 2015.

[34] S. Afifi, H. GholamHosseini and R. and Sinha, "FPGA Implementations of SVM Classifiers: A Review," *Springer Journal on SN Computer Science,* vol. 1, no. 133, pp. 1-17, April 2020.

[35] F. Lopes, J. Ferreira and M. Fernandes, "Parallel Implementation on FPGA of Support Vector Machines Using Stochastic Gradient Descent," in *Electronics 2019, 8, 631.*.

[36] K. Koliogeorgi, G. Zervakis, D. Anagnostos, K. Siozios and Z. N., "Optimizing SVM Classifier Through Approximate and High Level Synthesis Techniques," in *8th International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloni*, 2019.

[37] S. Afifi, H. GholamHosseini and R. Sinha, "A system on chip for melanoma detection using FPGA-based SVM classifier," *Microprocessors and Microsystems,* vol. 65, no. ISSN 0141-9331, pp. 57-68, 2019.

[38] D. H. Noronha, M. F. Torquato and M. A. Fernandes, "A parallel implementation of sequential minimal optimization on FPGA," *Microprocessors and Microsystems,* vol. 69, no. 0141-9331, pp. 138-151, 2019.

[39] V. Tsoutsouras, K. Koliogeorgi and S. Xydis, "An Exploration Framework for Efficient High-Level Synthesis of Support Vector Machines: Case Study on ECG Arrhythmia Detection for Xilinx Zynq SoC," in *J Sign Process Syst 88*, 2017.

[40] Struharik and V. Vranjkovic, "Coarse-grained reconfigurable hardware accelerator of machine learning classifiers," in *International Conference on Systems, Signals and Image Processing (IWSSIP)*, Bratislava, 2016.

[41] M. P. Bouganis, "A novel FPGA-based SVM classifier," in *International Conference on Field-Programmable Technology*, Beijing, 2010.

[42] C. Bouganis and M. Papadonikolakis, "A Heterogeneous FPGA Architecture for Support Vector Machine Training," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, Charlotte, NC, 2010, pp. 211-214, doi: 10.1109/FCCM*.

[43] B. Mandal, M. Sarma, K. Sarma and N. Mastorakis, "Implementation of systolic array based SVM classifier using multiplierless kernel," in *Proceedings of the 16th International Conference on Automatic Control, Modelling & Simulatio*, 2014.

[44] C. Bouganis and M. Papadonikolakis, "A scalable FPGA architecture for non-linear SVM training," in *2008 International Conference on Field-Programmable Technology, Taipei, 2008, pp. 337-340, doi: 10.1109/FPT.2008.4762412.*.

[45] M. Papadonikolakis, C. Bouganis and G. Constantinides, "Performance comparison of GPU and FPGA architectures for the SVM training problem," in *2009 International Conference on Field-Programmable Technology, Sydney, NSW, 2009, pp. 388-391, doi: 10.1109/FPT.20*.

[46] M. Pietron, "Comparison of GPU and FPGA Implementation of SVM Algorithm for Fast Image Segmentation," in *vol. 7767, 2013.*.

[47] Y. Yuan, K. Virupakshappa, Y. Jiang and E. Oruklu, "Comparison of GPU and FPGA based hardware platforms for ultrasonic flaw detection using support vector machines," in *2017 IEEE International Ultrasonics Symposium (IUS), Washington, DC, 2017, pp. 1-4*.

[48] S. Cadambi, "A Massively Parallel FPGA-Based Coprocessor for Support Vector Machines," in *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines, Napa, CA, 2009, pp. 115-122, doi: 10.1109/FCCM.2009.34.*.

[49] S. Wang, Y. Peng, G. Zhao and X. Peng, "Accelerating on-line training of LS-SVM with run-time reconfiguration," in *2011 International Conference on Field-Programmable Technology*, Dec. 2011.

[50] M. Paoletti, J. Haut, X. Tao, J. Miguel and A. Plaza, "A New GPU Implementation of Support Vector Machines for Fast Hyperspectral Image Classification," in *Remote Sens. 2020, 12, 1257.*.

[51] J. C. Porcello, "Designing and Implementing SVMs for High-Dimensional Knowledge Discovery Using FPGAs," in *2019 IEEE Aerospace Conference, Big Sky, MT, USA, 2019, pp. 1-8, doi: 10.1109/AERO.2019.8741916.*.

[52] H. Chen, "Hyperparameter Estimation in SVM with GPU Acceleration for Prediction of Protein-Protein Interactions," in *2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 2197-2204, doi: 10.1109/BigData47090.2019.9*.

[53] J. Sirkunan, N. Shaikh-Husin and M. N. Marsono, "Interleaved Incremental/Decremental Support Vector Machine for Embedded System," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 2019, pp. 1-5, doi: 10.1109/ISCAS.2019.870*.

[54] C. Kyrkou, T. Theocharides and C. Bouganis, "Boosting the Hardware-Efficiency of Cascade Support Vector Machines for Embedded Classification Applications,"

in *Int J Parallel Prog 46, 1220–1246 (2018). https://doi.org/10.1007/s10766-017-0514-1*.

[55] J. Shi, "Efficient Support Vector Machine Training Algorithm on GPUs," *AAAI*'18

[56] S. Afifi, "An Optimized Hardware System on Chip for a Support Vector Machine Classifier: a Case Study on Melanoma Detection," (2018)..

[57] I. Sayehi, M. Machhout and R. Tourki, "FPGA Implementation of SVM for Nonlinear Systems Regression," in *International Journal of Advanced Computer Science and Applications. 8. 10.14569/IJACSA.2017.080816.*, 2017.

[58] C. E. Santos, R. C. Sampaio, H. Ayala, S. Coelho, R. Jacobi and C. Llanos, "A SVM optimization tool and FPGA system architecture applied to NMPC," in *2017 30th Symposium on Integrated Circuits and Systems Design (SBCCI), Fortaleza, 2017, pp. 96-102.*.

[59] R. Saini, S. Saurav, D. C. Gupta and N. Sheoran, "Hardware implementation of SVM using system generator," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, 2017, pp. 2129-21*.

[60] J. Vanek, J. Michálek and J. Psutka, "A GPU-Architecture Optimized Hierarchical Decomposition Algorithm for Support Vector Machine Training," in *IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 12, pp. 3330-3343, 1 Dec. 2017, doi: 10.11*.

[61] S. Afifi, H. GholamHosseini and R. Sinha, "Hardware Acceleration of SVM-Based Classifier for Melanoma Images," in *Image and Video Technology -- PSIVT 2015 Workshops Springer International Publishing*, 2016.

[62] N. Z. Tarapore, D. B. Kulkarni and V. K. Prasad, "Implementation of parallel algorithm for support vector machine applied to intrusion detection systems," in *2016 International Conference on Computing, Analytics and Security Trends (CAST), Pune, 2016, pp. 17*.

[63] T. Li, X. Liu, Q. Dong, W. Ma and K. Wang, "HPSVM: Heterogeneous Parallel SVM with Factorization Based IPM Algorithm on CPU-GPU Cluster," in *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP), Heraklion*.

[64] P. Pouladzadeh, S. Shirmohammadi, A. Bakirov, A. Bulut and A. Yassine, "Cloud-based SVM for food categorization," in *Multimedia Tools Appl. 74, 14 (July 2015), 5243–5260. DOI:https://doi.org/10.1007/s11042-014-2116-x*.

[65] H. M. Hussain, K. Benkrid and H. Seker, "Dynamic partial reconfiguration implementation of the SVM/KNN multi-classifier on FPGA for bioinformatics application,," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Soci*.

[66] Y. You, H. Fu, S. L. Song, A. Randles, D. Kerbyson, A. Marquez, G. Yang and A. Hoisie, "Scaling Support Vector Machines on modern HPC platforms," *Journal of Parallel and Distributed Computing,,* vol. 76, pp. 16-31, 2015.

[67] C. Kyrkou, C. Bouganis, T. Theocharides and M. Polycarpou, "Embedded Hardware-Efficient Real-Time Classification With Cascade Support Vector Machines," in *IEEE Transactions on Neural Networks and Learning Systems, vol. 27, no. 1, pp. 99-112, Jan. 2016,*.

[68] S. Venkateshan, A. Patel and K. Varghese, "Hybrid Working Set Algorithm for SVM Learning With a Kernel Coprocessor on FPGA," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 10, pp. 2221-2232, Oct. 2015, doi: 10.1109/TVLSI*.

[69] X. Zhang and Z. Yifeng, "GPU Implementation of Parallel Support Vector Machine Algorithm with Applications to Intruder Detection," 2014.

[70] S. Song, H. Wang and L. Wang, "FPGA Implementation of a Support Vector Machine Based Classification System and Its Potential Application in Smart Grid,"

in *2014 11th International Conference on Information Technology: New Generations, Las Vegas, NV, 2014, pp*.

[71] B. Mandal, M. Sarma and K. Sarma, "Design of a systolic array based multiplierless support vector machine classifie," in *2014 International Conference on Signal Processing and Integrated Networks (SPIN), Noida, 2014, pp. 35-39, doi: 10.1109/SPIN.2014.*.

[72] L. Mohammed and A. Jallad, "Hardware Support Vector Machine (SVM) for satellite on-board applications, 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Leicester, 2014, pp. 256-261, doi: 10.1109/AHS.2014.6880185.".

[73] T. Li, H. Li, X. Liu, S. Zhang, K. Wang and Y. Yang, "GPU Acceleration of Interior Point Methods in Large Scale SVM Training," in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, VIC, 2013, pp*.

[74] J. Jin, X. Cai and X. Lin, "Efficient SVM Training Using Parallel Primal-Dual Interior Point Method on GPU," in *2013 International Conference on Parallel and Distributed Computing, Applications and Technologies, Taipei, 2013, pp. 12-17, doi: 10.1109/PDCAT.20*.

[75] C. Kyrkou, T. Theocharides and C. Bouganis, "An embedded hardware-efficient architecture for real-time cascade Support Vector Machine classification," in *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAM*.

[76] T. Groleat, M. Arzel and S. Vaton, "Hardware acceleration of SVM-based traffic classification on FPGA," in *8th International Wireless Communications and Mobile Computing Conference (IWCMC), Limassol, 2012, pp. 443-449, doi: 10.1109/IWCMC.2012.6314245.*, 2012.

[77] Z. Nie, "An FPGA Implementation of Multi-Class Support Vector Machine Classifier Based on Posterior Probability," 2012.

[78] A. Athanasopoulos, A. Dimou, V. Mezaris and I. Kompatsiaris, "GPU acceleration for support vector machines," 2011.

[79] D. Mahmoodi, A. Soleimani, H. Khosravi and M. Taghizadeh, "FPGA Simulation of Linear and Nonlinear Support Vector Machine," in *Journal of Software Engineering and Applications, Vol. 4 No. 5, 2011, pp. 320-328. doi: 10.4236/jsea.2011.45036.*.

[80] C. Kyrkou and T. Theocharides, "A Parallel Hardware Architecture for Real-Time Object Detection with Support Vector Machines," in *IEEE Transactions on Computers, vol. 61, no. 6, pp. 831-842, June 2012, doi: 10.1109/TC.2011.113.*.

[81] M. Ruiz-Llata, G. Guarnizo and M. Yébenes-Calvino, "FPGA implementation of a support vector machine for classification and regression," in *The 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona, 2010, pp. 1-5, doi: 10.1109/IJCNN.2010.5*.

[82] S. Bauer, S. Köhler, K. Doll and U. Brunsmann, "FPGA-GPU architecture for kernel SVM pedestrian detection," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, San Francisco, CA, 2010, pp. 61-68, doi: 10.1109/CVPR*.

[83] S. Chien and T. Lin, "Support Vector Machines on GPU with Sparse Matrix Format," in *2010 Ninth International Conference on Machine Learning and Applications, Washington, DC, 2010, pp. 313-318.*.

[84] K. Cao, H. Shen and H. Chen, "A parallel and scalable digital architecture for training support vector machines.," in *J. Zhejiang Univ. - Sci. C 11, 620–628 (2010). https://doi.org/10.1631/jzus.C0910500*.

[85] T. Theocharides and C. Kyrkou, "SCoPE: Towards a Systolic Array for SVM Object Detection," in *IEEE Embedded Systems Letters, vol. 1, no. 2, pp. 46-49, Aug. 2009, doi: 10.1109/LES.2009.2034709.*.

[86] J. Manikandan, B. Venkataramani and V. Avanthi, "FPGA Implementation of Support Vector Machine Based Isolated Digit Recognition System," in *2009 22nd International Conference on VLSI Design, New Delhi, 2009, pp. 347-352, doi: 10.1109/VLSI.Design.2009.23.*.

[87] E. Alba, D. Anguita, A. Ghio and S. Ridella, "Using Variable Neighborhood Search to improve the Support Vector Machine performance in embedded automotive applications," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Co*.

[88] D. Anguita, A. Ghio, S. Pischiutta and S. Ridella, "A Hardware-friendly Support Vector Machine for Embedded Automotive Applications," in *2007 International Joint Conference on Neural Networks, Orlando, FL, 2007, pp. 1360-1364, doi: 10.1109/IJCNN.2007.4371156*.

[89] P.-H. Chen, R.-E. Fan and C.-J. Lin, "A study on SMO-type decomposition methods for support vector machines," *IEEE Transactions on Neural Networks,* vol. 17, no. 4, pp. 893-908, 2006.

[90] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks,* vol. 13, pp. 415-425, 2002.

[91] S. S. Keerthi and E. G. Gilbert, "Convergence of a Generalized SMO Algorithm for SVM Classifier Design," *Machine Learning,* vol. 46, no. 1, pp. 351-360, 2002.

[92] V. Vapnik, S. E. Golowich and A. Smola, "Support Vector Method for Function Approximation, Regression Estimation and Signal Processing," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, Denver, Colorado, 1996.

[93] B. Schölkopf and A. J. Smola, Learning with kernels: support vector machines, regularization, optimization, and beyond, Cambridge, Mass: MIT Press, 2002.

[94] B. Schölkopf, C. J. C. Burges and A. J. Smola, Advances in Kernel Methods: Support Vector Learning, Cambridge, MA: MIT Press, 1999.

[95] J. Platt, "Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines," April, 1998.

[96] C.-C. Chang, C.-W. Hsu and C.-J. Lin, "The analysis of decomposition methods for support vector machines," *IEEE Transactions on Neural Networks,* vol. 11, no. 4, pp. 1003-1008, July 2000.

[97] S. Lucidi, L. Palagi, A. Risi and M. Sciandrone, "A convergent decomposition algorithm for support vector machines," *Computational Optimization and Applications,* vol. 38, no. 2, pp. 217-234, 2007.

[98] M. Andersen, J. Dahl and L. Vandenberghe, "CVXOPT: A python package for convex optimization v.1.2.2," 2012. [Online]. Available: https://cvxopt.org/index.html.

[99] C.-J. Lin, "A formal analysis of stopping criteria of decomposition methods for support vector machines," *IEEE Transactions on Neural Networks,* vol. 13, no. 5, pp. 1045-1052, 2002.

[100] P. Walton, "Artificial Intelligence and the Limitations of Information," *Information,* vol. 9, p. 332, December 2018.

[101] O. Obulesu, M. Mahendra and M. ThrilokReddy, "Machine Learning Techniques and Tools: A Survey," in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA*, 2018.

[102] "Getting Started with the Virtex-6 FPGA ML605 Embedded Kit," San Jose, 2010.

[103] Xilinx, Inc., "Xilinx Virtex-6 FPGAs," Xilinx, [Online]. Available: https://www.xilinx.com/support/documentation/selection-guides/virtex6-product-table.pdf.

[104] Xilinx Inc., "ML605 Hardware User Guide," San Jose, 2019.

[105] Xilinx, Inc., "Virtex-6 Family Overview - DS150 (v2.5)," Aug 2015. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds150.pdf.

[106] Xilinx, "Virtex-6 FPGA Memory Resources - User Guide UG363 (v1.8)," Feb 2014. https://www.xilinx.com/support/documentation/user_guides/ug363.pdf.

[107] D. Lilja and S. Sapatnekar, Designing Digital Computer Systems with Verilog, Minneapolis: Cambridge University Press, 2004.

[108] Xilinx, "Virtex-6 Libraries Guide for HDL Designs," Xilinx, San Jose, CA, 2013.

[109] Xilinx, Inc., "Xilinx Power Estimator User Guide," Xilinx, San Jose, CA, 2013.

[110] Xilinx, Inc., "Xilinx Power Tools," Xilinx, Inc., San Jose, CA, 2013.

[111] Xilinx, "Virtex-6 Libraries Guide for Schematic Designs," Xilinx, San Jose CA.

[112] Xilinx, Inc., "ISE Design Suite 14: Release Notes, Installation, and Licensing," Xilinx, Inc., San Jose, CA, 2013.

[113] Xilinx, Inc., "ISE In-Depth Tutorial," Xilinx, Inc., San Jose, CA, 2009.

[114] Xilinx, Inc., "ISE Tutorial," Xilinx, Inc., San Jose, CA, 2013.

[115] Xilinx, Inc., "ISim User Guide," Xilinx, Inc., San Jose, CA.

[116] Xilinx Inc., "AXI Interface Based ML605/SP605 Microblaze Processor Subsystem," San Jose, 2012.

[117] Xilinx Inc., "Vivado Design Suite User Guide," Xilinx, San Jose, 2017.

[118] Xilinx, Inc., "ChipScope Pro Software and Cores," Xilinx, San Jose, CA, 2012.

[119] Xilinx, Inc., "Command Line Tools User Guide," Xilinx, Inc., San Jose, CA, 2013.

[120] Xilinx, Inc., "LogiCORE IP Product Guide," 05 October 2016. [Online]. https://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v2_0/pg079-axi-timer.pdf.

[121] D. Dua and C. Graff, "UCI - Machine Learning Repository," University of California, Irvine, School of Information and Computer Sciences, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.

[122] V. Sigillito, "Ionosphere Data Set Machine Learning Repository," University of California, Irvine, School of Information and Computer Sciences, 1989. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/ionosphere.

[123] Xilinx Inc., "ML605 Reference Design User Guide," 2009.

[124] Xilinx, "Getting Started with the Virtex-6 FPGA ML605 Embedded Kit," 2011.

[125] Xilinx, Inc., "Constraints Guide," Xilinx, San Jose, CA, 2013.

[126] Xilinx Inc., "AXI Reference Guide - UG761 (v13.1)," March 2011. [Online]. https://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf.

[127] S.N. Shahrouzi and D.G. Perera, "Optimized Hardware Accelerators for Data Mining Applications on Embedded Platform: Case Study Principal Component Analysis," Elsevier Journal on Microprocessor and Microsystems (MICPRO), vol. 65, pp. 79-96, March 2019.

[128] D.G. Perera and K.F. Li, "Embedded Hardware Solution for Principal Component Analysis," in Proceedings of IEEE Pacific Rim International Conference on Communication, Computers, and Signal Processing, (PacRim'11), pp.730-735, Victoria, BC, Canada, August 2011.

[129] A. Abdelhadi and G. Lemieux, "A Multi-ported Memory Compiler," in *Proc. of 24th IEEE Annual International*, 2016.

[130] D. Craigen, "“Embedded Systems”, Chapter 2 from “Validation, Verification and Certification of Embedded Systems”, N," NATO Research and Technology Organization, Oct 2005.

[131] D.G. Perera and K.F. Li, "Analysis of Single-Chip Hardware Support for Mobile and Embedded Applications," in Proceedings of IEEE Pacific Rim International Conference on Communication, Computers, and Signal Processing, (PacRim’13), pp. 369-376, Victoria, BC, Canada, August 2013.

[132] F. Pedregosa and G. Varoquaux, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[133] N. Peter, S. Ateeq Ur Rahman, G. Rakesh and H. Erik, "Hardware implementation of the exponential function using Taylor series," in *2014 NORCHIP*, Oct 2014.

[134] H. T. Bui, "Design and Synthesis of an IEEE-754 Exponenetial Function," in *Proceedings of the 1999 IEEE Canadian Conference on Electrical and Computer Engineering*, Alberta, Canada, May 1999.

[135] Y. Ago, K. Nakano and Y. Ito, "A Classification Processor for a Support Vector Machine with Embedded DSP Slices and Block RAMs in the FPGA," in *Proc. of 7th IEEE International Symposium on Embedded Multicore/Manycore System-on-Chip (MCSoC)*, 2013.

[136] Xilinx, Inc., "LogiCORE IP Floating-Point Operator - PG060 (v7.0)," April 2014. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/floating_point/v7_0/pg060-floating-point.pdf.

[137] X.-S. Yang, Introduction to Mathematical Optimization, Cambridge UK: Cambridge International Science Publishing, 2008.

[138] R. Struharik and V. Vranjkovic, "Coarse-grained reconfigurable hardware accelerator of machine learning classifiers," in *2016 International Conference on Systems, Signals and Image Processing (IWSSIP), Bratislava, 2016, pp. 1-5, doi: 10.1109/IWSSIP.2016.7502*.

[139] B. Mandal, M. P. Sarma and K. K. Sarma, "Design of a systolic array based multiplierless support vector machine classifier," in *2014 International Conference on Signal Processing and Integrated Networks (SPIN), Noida, 2014, pp. 35-39, doi: 10.1109/SPIN.2014.*.

[140] I. Amezzane, Y. Fakhri, M. El Aroussi and M. Bakhouya, "Hardware Acceleration of SVM Training for Real-Time Embedded Systems: Overview," in *Recent Advances in Mathematics and Technology: Proceedings of the First International Conference on Technology, Engineering, and Mathematics, Kenitra, Morocco, March 26-27, 2018*, Springer International Publishing, 2020, pp. 131--139.

[141] S. Afifi, "Hardware Implementations of SVM on FPGA: A State-of-the-Art Review of Current Practice.," (2015)..

[142] S. Tavara., "Parallel Computing of Support Vector Machines: A Survey.," in *ACM Comput. Surv. 51, 6, Article 123 (February 2019), 38 pages. DOI:https://doi.org/10.1145/3280989*, 2019.

[143] Xilinx, Inc., "Xilinx ML605 Hardware User Guide (UG534)," Xilinx, Inc., 26 February 2019. [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf.

[144] S. Shahrouzi and D. Perera, *User Guides for Xilinx ISE Design Suite, FPGA-Based Embedded Systems Designs, and Design Flow and Methodologies,* Colorado: Electrical and Computer Engineering Department, University of Colorado, Colorado Springs, 2015, p. 382.

[145] Z. Navabi, Verilog Digital System Design, New York: McGraw-Hill, 2006.

[146] Xilinx, "RTL Technology and Schematic Viewers," Xilinx, San Jose, CA, 2009.

[147] Xilinx, Inc., "Synthesis and Simulation Design Guide," Xilinx, San Jose, CA.

[148] Xilinx, Inc., "Timing Closure User Guide," Xilinx, San Jose, CA, 2012.

[149] Xilinx, Inc., "PlanAhead User Guide," Xilinx, Inc., San Jose, CA, 2013.

[150] Xilinx, Inc., "Power Methodology Guide," Xilinx, Inc., San Jose, CA, 2013.

[151] Xilinx, "RTL and Technology Schematic Viewers," San Jose, CA, 2012.

[152] Xilinx, Inc., "Xilinx/Cadence PCB Guide," Xilinx, Inc., San Jose, CA, 2011.

[153] Xilinx, Inc., "Xilinx/Mentor Graphics PCB Guide," San Jose, CA, 2011.

[154] Xilinx, Inc., "XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices," Xilinx, Inc., San Jose, CA, 2013.

[155] Xilinx, Inc., "LogiCORE IP AXI Master Burst DS844 (v1.00.a)," June 2011. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_master_burst/v1_00_a/ds844_axi_master_burst.pdf.

[156] M. Alom, T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, B. Van Essen, A. Awwal and V. Asari, *A State-of-the-Art Survey on Deep Learning Theory and Architectures,* 2019.

[157] W. L. Alsaadi, Z. Wang, X. Liu, N. Zeng, Y. Liu and F. E., "A survey of deep neural network architectures and their applications," *Neurocomputing,* vol. 234, no. 0925-2312, pp. 11-26, 2017.

[158] O. Abiodun, A. Jantan, O. Omolara, K. Dada, N. Mohamed and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon,* vol. 4, no. 11, p. e00938, 2018.

[159] Z. L. Liu, W. Yang, S. Peng and Fan, *A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,* https://arxiv.org/abs/2004.02806, 2020.

[160] Q. Zhang, L. T. Yang, Z. Chen and P. Li, "A survey on deep learning for big data," *Information Fusion,,* vol. 42, pp. 146-157, 2018.

[161] S. Grigorescu, B. Trasnea, T. Cocias and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of field robotics,* vol. 37, no. 3, pp. 362-386, 2019.

[162] Q. Mao, F. Hu and Q. Hao, "Deep Learning for Intelligent Wireless Networks: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials,* vol. 20, no. 4, pp. 2595-2621, 2018.

[163] U.S. Department of Energy, "United States Data Center Energy Usage Report," June 2016. [Online]. Available: https://www.osti.gov/servlets/purl/1372902#:~:text=Based%20on%20current%20trend%20estimates,73%20billion%20kWh%20in%202020..

[164] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai and T. Chen, "Recent advances in convolutional neural networks," *Pattern Recognition,* vol. 77, no. 0031-3203, pp. 354-377, 2018.

[165] A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *Artificial Intelligence Review,* vol. 53, no. 8, 2020.

[166] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy and B. Hodjat, "Chapter 15 - Evolving Deep Neural Networks," *Artificial Intelligence in the Age of Neural Networks and Brain Computing,* no. 9780128154809, pp. 293-312, 2019.

[167] Y. Chen, Y. Xie, L. Song, F. Chen and T. Tang, "A Survey of Accelerator Architectures for Deep Neural Networks," *Engineering,* vol. 6, pp. 264-274, 2020.

[168] A. Shawahna & et al., "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review," in *IEEE Access*, 2019.

[169] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Computing and Applications,* vol. 32, pp. 1109-1139, 01 02'20.

[170] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie and X. Zhou, "DLAU: A Scalable Deep Learning Accelerator Unit on FPGA," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

[171] Y. Ma, N. Suda, Y. Cao, S. Vrudhula and J.-s. Seo, "ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler," in *Integration'18*.

[172] M. Alawad and M. lin, "Scalable FPGA Accelerator for Deep Convolutional Neural Networks with Stochastic Streaming," in *IEEE Transactions on Multi-Scale Computing Systems*, 2018.

[173] T. Tsai, Y. Ho and M. Sheu, "Implementation of FPGA-based Accelerator for Deep Neural Networks," in *IEEE 22nd International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, Romania, 2019.

[174] S. Srinivasan, "High Performance Scalable FPGA Accelerator for Deep Neural Networks," *ArXiv abs/1908.11809,* 2019.

[175] T. V. Huynh, "Deep neural network accelerator based on FPGA," in *4th NAFOSTED Conference on Information and Computer Science*, Hanoi, 2017.

[176] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou and Y. Xu, "Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks," *ACM Trans. Reconfigurable Technol. Systems,* vol. 10, no. 3, p. 23, July 2017.

[177] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for FPGA-based convolutional neural networks," in *27th International Conference on Field Programmable Logic and Applications (FPL)*, Ghent, 2017.

[178] Y. Shen, M. Ferdman and P. Milder, "Maximizing CNN Accelerator Efficiency Through Resource Partitioning," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, Toronto, ON, Canada, 2017.

[179] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang and J. Cong, "Automated Systolic Array Architecture Synthesis for High Throughput CNN Inference on FPGAs," in *Proceedings of the 54th Annual Design Automation Conference 2017*, Austin, TX, USA, 2017.

[180] Y. Ma, Y. Cao, S. Vrudhula and J.-s. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *ACM/SIGDA International Symposium on*, Monterey, California, USA, 2017.

[181] L. Lu, Y. Liang, Q. Xiao and S. Yan, "Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs," in *IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, '17.

[182] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto and H. P. Graf, "A Massively Parallel Coprocessor for Convolutional Neural Networks," in *20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2009.

[183] S. Chakradhar, M. Sankaradas, V. Jakkula and S. Cadambi, "A Dynamically Configurable Coprocessor for Convolutional Neural Networks," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, France, '10.

[184] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello and Y. LeCun, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *CVPR 2011 WORKSHOPS*, Colorado Springs, CO, 2011.

[185] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss and E. Chung, "Accelerating Deep Convolutional Neural Networks Using Specialized Hardware," in *Microsoft Research*, https://www.microsoft.com/en-us/research/publication/accelerating-deep-convolutional-neural-networks-using-specialized-hardware/, 2015.

[186] Xilinx Inc., "Getting Started with the Xilinx Virtex-6 FPGA ML605 Evaluation Kit," San Jose, 2011.

[187] InAccel.com, "CPU, GPU or FPGA: A use case on Logistic regression training in cloud computing platforms," InAccel, 13 December 2019. [Online]. Available: https://inaccel.com/cpu-gpu-or-fpga-performance-evaluation-of-cloud-computing-platforms-for-machine-learning-training/.

[188] Aldec.com, "FPGA vs GPU for Machine Learning Applications: Which one is better?," ALDEC - The Design Verification Company, [Online]. Available: https://www.aldec.com/en/company/blog/167--fpgas-vs-gpus-for-machine-learning-applications-which-one-is-better.

[189] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu and S. Zhang., "Understanding Performance Differences of FPGAs and GPUs," in *In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '18)*, Association for Computing Machinery, New York, NY, USA, 288, 2018.

[190] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. O. G. Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra and G. Boudoukh, "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?," in *In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17).*, Association for Computing Machinery, New York, NY, USA, 5–14., 2018.

[191] N. Jouppi, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, Toronto, ON, Canada, 2017 pp. 1-12., 2017.

[192] A. Reuther, P. Michaleas, M. Jones, V. Gadepally and S. S. a. J. Kepner, "Survey and Benchmarking of Machine Learning Accelerators," in *IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham MA, USA, 2019.

[193] Berten - Digital Signal Processing, "GPU vs FPGA Performance Comparison," 2016. [Online]. Available:

http://www.bertendsp.com/pdf/whitepaper/BWP001_GPU_vs_FPGA_Performance_Comparison_v1.0.pdf.

[194] A. Stevens, "Nvidia DGX-2 review: More AI bang, for a lot more bucks," ZDNet, 04 January 2019. [Online]. Available: https://www.zdnet.com/article/nvidia-dgx-2-review-more-ai-bang-for-a-lot-more-bucks/#:~:text=For%20%24400%2C000%2C%20you%20could%20get,the%20most%20complex%20AI%20challenges%22..

[195] Arrow.com, "FPGA vs CPU vs GPU vs Microcontroller: How Do They Fit into the Processing Jigsaw Puzzle?," Arrow, 05 October 2018. [Online]. Available: https://www.arrow.com/en/research-and-events/articles/fpga-vs-cpu-vs-gpu-vs-microcontroller.

[196] B. Newman, "NVIDIA Tesla V100 Price Analysis," Microway, 08 May 2018. [Online]. Available: https://www.microway.com/hpc-tech-tips/nvidia-tesla-v100-price-analysis/.

[197] A. L. Caterini and D. E. Chang, Deep Neural Networks in a Mathematical Framework, Springer Publishing Company, Incorporated., 2018.

[198] Xilinx Inc., "Virtex-6 Family Overview - DS150 (v2.5)," Aug 2015. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds150.pdf.

[199] Xilinx Inc., "LogiCORE IP AXI Master Burst DS844 (v1.00.a)," June 2011. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_master_burst/v1_00_a/ds844_axi_master_burst.pdf.

[200] Xilinx Inc., "Virtex-6 FPGA Memory Resources - User Guide UG363 (v1.8)," Feb 2014. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug363.pdf.

[201] S. Raschka, Python Machine Learning, BirmingHam - Mumbai: Packt Publishing, 2015.

[202] D.G. Perera and Kin F. Li, "Analysis of Computation Models and Application Characteristics Suitable for Reconfigurable FPGAs", in Proceedings of the 10th IEEE International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing, (3PGCIC'15), pp. 244-247, Krakow, Poland, November 2015.

[203] D.G. Perera and Kin F. Li, "Hardware Acceleration for Similarity Computations of Feature Vectors," IEEE Canadian Journal of Electrical and Computer Engineering, (CJECE), vol. 33, no. 1, pp. 21-30, Winter 2008.

[204] D.G. Perera and Kin F. Li, "FPGA-Based Reconfigurable Hardware for Compute Intensive Data Mining Applications", in Proceedings of 6th IEEE International Conference on P2P, Parallel, Grid, Cloud, and Internet Computing, (3PGCIC'11), pp. 100-108, Barcelona, Spain, October 2011.

[205] D.G. Perera and K.F. Li, "On-Chip Hardware Support for Similarity Measures," in Proceedings of IEEE Pacific Rim International Conference on Communication, Computers, and Signal Processing, (PacRim'07), pp. 354-358, Victoria, BC, Canada, August 2007.

[206] K.F. Li and D.G. Perera, "An Investigation of Chip-Level Hardware Support for Web Mining," in Proceedings of IEEE International Symposium on Data Mining and Information Retrieval, (DMIR'07), pp. 341-348, Niagara Falls, ON, Canada, May 2007.

[207] K.F. Li and D.G. Perera, "A Hardware Collective Intelligent Agent", Transactions on Computational Collective Intelligence, LNCS 7776, Springer, pp. 45-59, 2013.

[208] A.K. Madsen and D.G. Perera, "Efficient Embedded Architectures for Model Predictive Controller for Battery Cell Management in Electric Vehicles", EURASIP Journal on Embedded Systems, SpringerOpen, vol. 2018, article no. 2, 36-page manuscript, July 2018.

[209] A.K. Madsen, M.S. Trimboli, and D.G. Perera, "An Optimized FPGA-Based Hardware Accelerator for Physics-Based EKF for Battery Cell Management", in

Proceedings of the IEEE International Symposium on Circuits and Systems, (ISCAS'20), 5-page manuscript, Seville, Spain, May 2020.

[210] A.K. Madsen, "Optimized Embedded Architectures for Model Predictive Control Algorithms for Battery Cell Management Systems in Electric Vehicles"; PhD Dissertation, Department of Electrical & Computer Engineering, University of Colorado Colorado Springs, August 2020.

[211] A. Alkamil and D.G. Perera, "Efficient FPGA-Based Reconfigurable Accelerators for SIMON Cryptographic Algorithm on Embedded Platforms", in Proceedings of the IEEE International Conferences on Reconfigurable Computing and FPGAs, (ReConFig'19), 8-page manuscript, Cancun, Mexico, December 2019.

[212] A. Alkamil and D.G. Perera, "Towards Dynamic and Partial Reconfigurable Hardware Architectures for Cryptographic Algorithms on Embedded Devices", IEEE Access, Open Access Journal in IEEE, vol. 8, pp: 221720 – 221742, 10th December 2020.

[213] A. Alkamil, "Dynamic Reconfigurable Cryptographic Architectures to Improve Performance and Security on Embedded Systems", PhD Dissertation, Department of Electrical & Computer Engineering, University of Colorado Colorado Springs, 5th February 2021.

[214] M.A. Mohsin and D.G. Perera, "An FPGA-Based Hardware Accelerator for K-Nearest Neighbor Classification for Machine Learning on Mobile Devices", in Proceedings of the IEEE/ACM International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, (HEART'18), 6-page manuscript, Toronto, Canada, June 2018.

[215] S. Ramadurgam and D.G. Perera, "Composing Optimized FPGA-Based Hardware Architectures for Machine Learning Applications on Embedded Devices: Case Study Convex Optimization-Based SVM", Submitted to the Elsevier Journal of Microelectronics (MEJ), 32-page manuscript, submitted on 2nd February 2021.

[216] J.P. Miro, " FPGA-Based Accelerators for Convolutional Neural Networks on Embedded Devices", MSc Thesis, Department of Electrical & Computer Engineering, University of Colorado Colorado Springs, CO, USA, May 2020.

[217] M.A. Mohsin, "An FPGA-Based Hardware Accelerator for K-Nearest Neighbor Classification for Machine Learning", MSc Thesis, Department of Electrical & Computer Engineering, University of Colorado Colorado Springs, CO, USA, December 2017.

[218] L.H. Garcia, "An FPGA-Based Hardware Accelerator for Sequence Alignment by Genetic Algorithm", MSc Thesis, Department of Electrical & Computer Engineering, University of Colorado Colorado Springs, CO, USA, December 2019.

[219] R. Raghavan and D.G. Perera, "A Fast and Scalable FPGA-Based Parallel Processing Architecture for K-Means Clustering for Big Data Analysis", in Proceedings of the IEEE Pacific Rim International Conference on Communications, Computers, and Signal Processing, (PacRim'17), pp. 1-8, Victoria, BC, Canada, August 2017.

[220] D.G. Perera and Kin F. Li, "Parallel Computation of Similarity Measures Using an FPGA-Based Processor Array," in Proceedings of 22nd IEEE International Conference on Advanced Information Networking and Applications, (AINA'08), pp. 955-962, Okinawa, Japan, March 2008.

[221] R. Raghavan, "A Fast and Scalable Hardware Architecture for K-Means Clustering for Big Data Analysis", MSc Thesis, Department of Electrical & Computer Engineering, University of Colorado Colorado Springs, CO, USA, May 2016.

[222] D.G. Perera and K.F. Li, "A Design Methodology for Mobile and Embedded Applications on FPGA-Based Dynamic Reconfigurable Hardware", International Journal of Embedded Systems, (IJES), Inderscience publishers, 23-page manuscript, vol. 11, no. 5, September 2019.

[223] D.G. Perera, “Analysis of FPGA-Based Reconfiguration Methods for Mobile and Embedded Applications”, in Proceedings of 12th ACM FPGAWorld International Conference, (FPGAWorld’15), pp. 15-20, Stockholm, Sweden, September 2015.

[224] S.N Shahrouzi and D.G. Perera, “HDL Code Optimization: Impact on Hardware Implementations and CAD Tools”, in Proceedings of the IEEE Pacific Rim International Conference on Communications, Computers, and Signal Processing, (PacRim’19), 9-page manuscript, Victoria, BC, Canada, August 2019.

[225] D.G. Perera, “Chip-Level and Reconfigurable Hardware for Data Mining Applications,” PhD Dissertation, Department of Electrical & Computer Engineering, University of Victoria, Victoria, BC, Canada, April 2012.

[226] S.N. Shahrouzi, "Optimized Embedded and Reconfigurable Hardware Architectures and Techniques for Data Mining Applications on Mobile Devices", PhD Dissertation, Department of Electrical & Computer Engineering, University of Colorado Colorado Springs, December 2018.

[227] D.G. Perera and Kin F. Li, “Similarity Computation Using Reconfigurable Embedded Hardware,” in Proceedings of 8th IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC’09), pp. 323-329, Chengdu, China, December 2009.

[228] S.N. Shahrouzi and D.G. Perera, “Dynamic Partial Reconfigurable Hardware Architecture for Principal Component Analysis on Mobile and Embedded Devices”, EURASIP Journal on Embedded Systems, SpringerOpen, vol. 2017, article no. 25, 18-page manuscript, 21st February 2017.

[229] D.G. Perera, “Reconfigurable Architectures for Data Analytics on Next-Generation Edge-Computing Platforms”, Featured Article, IEEE Canadian Review, vol. 33, no. 1, Spring 2021.

[230] S.N. Shahrouzi, A. Alkamil, and D.G. Perera, "Towards Composing Optimized Bi-Directional Multi-Ported Memories for Next-Generation FPGAs", IEEE Access, Open Access Journal in IEEE, vol. 8, no. 1, pp. 91531-91545, 14th May 2020.

[231] S.N. Shahrouzi and D.G. Perera, "An Efficient Embedded Multi-Ported Memory Architecture for Next-Generation FPGAs", in Proceedings of 28th Annual IEEE International Conferences on Application-Specific Systems, Architectures, and Processors, (ASAP'17), pp. 83-90, Seattle, WA, USA, July 2017.

[232] S.N. Shahrouzi and D.G. Perera, "An Efficient FPGA-Based Memory Architecture for Compute-Intensive Applications on Embedded Devices", in Proceedings of the IEEE Pacific Rim International Conference on Communications, Computers, and Signal Processing, (PacRim'17), pp. 1-8, Victoria, BC, Canada, August 2017.

[233] S.N. Shahrouzi and D.G. Perera, "Optimized Counter-Based Multi-Ported Memory Architectures for Next-Generation FPGAs", in Proceedings of the 31st IEEE International Systems-On-Chip Conference, (SOCC'18), pp. 106-111, Arlington, VA, Sep. 2018.

APPENDIX A

A.1 Support Vector Machines – Scikit Learn

Scikit-learn [132] is a free python machine learning library [201], which provides simple, efficient tools for classification, regression, clustering, dimensionality reduction, model selection and preprocessing. Scikit-learn library is built using the open-source libraries such as numPy, SciPy, Matplotlib, which are easier for the user to modify the existing programs.

For Classification, various types of supervised and semi-supervised algorithms are available, among which the following command serves as a simple example to illustrate utilization of the tool for support vector machines.

class sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

Table 29. List of parameters for support vector machine in Scikit learn

Serial	Parameter	Default	Required	Data type	Usage
1	C	1	Optional	Float	Soft margin
2	kernel	rbf	Optional	String	If kernel is selected
3	degree	3	Optional	int	If poly kernel is selected
4	gamma	auto	optional	float	
5	coef0	0	optional	float	It is only significant in poly and sigmoid
6	Shrinking	TRUE	optional	Boolean	
7	Probability	FALSE	optional	Boolean	
8	tol	1.00E-03	optional	Float	
9	cache_size		optional	Float	If kernel is used
10	class_weight	None	optional	{dict, 'balanced'}	
11	verbose	FALSE		boolean	
12	max_iter	-1	Optional	int	
13	decision_function_shape	ovr			
14	random_state	None	Optional	int	
15	Step size	0.01	Optional	float	

Table A.1 lists all the possible parameters available to modify a classifier suitable for specific applications. Some of the parameters such as C: is a penalty parameters used to limit the misclassification rate by adding a penalty to the classifier. Another parameter called kernel provides different mathematical kernel options such as linear, polynomial, Gaussian kernel, sigmoid and also custom kernels. Table A.2 lists additional functions available to train a support vector machine classifier.

Table 30. List of additional functions for support vector machine in Scikit learn

Attributes usage	**Description**	
clf.support_vectors	*# get support vectors*	array-like, shape = [n_SV, n_features]
clf.support_	*# get indices of support vectors*	array-like, shape = [n_SV]
clf.n_support_	*# get number of support vectors for each class*	array-like, dtype=int32, shape = [n_class]
clf.dual_coef_	Coefficients of the support vector in the decision function. For multiclass, coefficient for all 1-vs-1 classifiers. The layout of the coefficients in the multiclass case is somewhat non-trivial. See the section about multi-class classification in the SVM section of the User Guide for details.	array, shape = [n_class-1, n_SV]
clf.coef_	Weights assigned to the features (coefficients in the primal problem). This is only available in the case of a linear kernel. coef_ is a readonly property derived from dual_coef_ and support_vectors_.	array, shape = [n_class-1, n_features]
clf.intercept_	Constants in decision function.	array, shape = [n_class * (n_class-1) / 2]

Table 31. Code example for support vector machine in Scikit learn

Examples [132]

```
from sklearn import svm
import numpy as np
import matplotlib as mplt
X = [[0, 0], [1, 1]]
y = [0, 1]
clf = svm.SVC()
clf.fit(X, y)
print(clf.support_vectors)
```

A.2 Datasets for Classification

Details of the datasets [122] used to evaluate the classifier are as follows:

Table 32. Wisconsin Breast Cancer diagnostic datasets

Total of Samples	569
Dataset	Wisconsin Cancer Dataset
Samples per class	212 (M), 357 (B)
Total of Samples	569
Dimensionality	30
Features	Real, Positive
Link	https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html#sklearn.datasets.load_breast_cancer
Download Dataset	https://goo.gl/U2Uwz2
Stage I	Kernel Computations
Stage II	Convex Optimization
Stage III	Testing
Total	Summation of the overall execution time for the above stages
Accuracy (%)	(Number of correct classified data / total number of testing vectors) * 100
Speedup	Hardware execution time / Software execution time
Linear	$\mathcal{K}(xi,xj) = xi.xj$
Poly	$\mathcal{K}(xi,xj) = (\gamma + xi.xj)^d$
Gaussian	$\mathcal{K}(xi,xj) = e^{(-(\lvert xi-xj\rvert)^2)/2\sigma^2}$
	d: degree, γ: Coef0
C: Penalty parameter	Affects all kernel, it is dependent on the objective function, but not on the kernel function
# sv	Total number of support vector out of # of training samples
Windows 7 Home premium	i7-3610QM @ 2.3GHz, 8GB RAM, 64-bit OS
Micro-Blaze	32-bit soft processor, 100MHz, 128kB BRAM
Hardware module	100MHz

Table 33. Wisconsin Breast Cancer diagnostic datasets – Data size

Training Set				
Percentage	**# train samples**	**# test samples**	**Train Data Size**	**Test Data Size**
10%	57	512	54720	491520
20%	114	455	109440	436800
30%	171	398	164160	382080
40%	228	341	218880	327360
50%	285	284	273600	272640
60%	342	227	328320	217920
70%	399	170	383040	163200
80%	456	113	437760	108480
90%	513	56	492480	53760

Table 34. Ionosphere datasets

Total of Samples	351
Dataset	Ionosphere Dataset
Samples per class	(b), (g)
Total of Samples	351
Dimensionality	34
Features	Real, Integer

Table 35. Ionosphere datasets – Data size

Training Set				
Percentage	**# train samples**	**# test samples**	**Train Data Size**	**Test Data Size**
10%	36	315	34560	302400
20%	71	280	68160	268800
30%	106	245	101760	235200
40%	141	210	135360	201600
50%	176	175	168960	168000
60%	211	140	202560	134400
70%	246	105	236160	100800
80%	281	70	269760	67200
90%	316	35	303360	33600