M.Tech. Thesis

PERFORMANCE SPEEDUP OF FUZZY LOGIC SYSTEMS USING GPU AND CUDA

Submitted in partial fulfillment of the requirements for the degree of Master of Technology in Electronics & Communication Engineering

by

Durlabh Chauhan (1269835)

Under the Supervision of

Dr. Satvir Singh

PUNJAB TECHNICAL UNIVERSITY

Jalandhar-Kapurthala Highway, Jalandhar

SHAHEED BHAGAT SINGH
STATE TECHNICAL CAMPUS
Moga Road (NH-95), Ferozepur-152004 (PB) INDIA

October 2014

CERTIFICATE

I, **Durlabh Chauhan (1269835)**, hereby declare that the work being presented in this thesis on PERFORMANCE SPEEDUP OF FUZZY LOGIC SYSTEMS USING GPU AND CUDA is an authentic record of my own work carried out by me during my course under the supervision of Dr. Satvir Singh. This is submitted to the Department of ECE at Shaheed Bhagat Singh State Technical Campus, Ferozepur (affiliated to Punjab Technical University, Jalandhar) as partial fulfillment of requirements for award of the degree of Master of Technology in Electronics & Communication Engineering.

Durlabh Chauhan (1269835)

To the best of my knowledge, this thesis has not been submitted to Punjab Technical University, Jalandhar or to any other university or institute for award of any other degree or diploma. It is further understood that by this certificate, the undersigned do/does not endorse or approve any statement made, opinion expressed or conclusion drawn therein, however, approve the thesis only for the purpose for which it is submitted.

Dr. Satvir Singh [Supervisor]

The M.Tech Viva-Voce Examination of Durlabh Chauhan (1269835) is held at Department of ECE, SBS State Technical Campus, Ferozepur on .....

External Examiner
Name: .....

Sanjeev Dewra
Head, Department of ECE

TEN LIFE LESSONS

1. Follow Your Curiosity
2. Perseverance in Priceless
3. Focus on the Present
4. The Imagination is Powerful
5. Make Mistakes
6. Live in the Moment
7. Create Value
8. Don't be Repetitive
9. Knowledge comes from Experience
10. Learn the Rules and Then Play Better

- Albert Einstein

Dedicated to

My Parents

ACKNOWLEDGEMENTS

Apart from the efforts of myself, the success of this report depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this project.

I would like to express the deepest appreciation and gratitude to my supervisor, **Dr. Satvir Singh**, Professor, Department of Electronics & Communication Engineering, SBS State Technical Campus, Ferozepur (Punjab), India, who has the attitude and the substance of a genius. He continually and convincingly conveyed a spirit of adventure in regard to research and scholarship, and an excitement in regard to teaching. Without his guidance and persistent help this report would not have been possible. I can't say *Thank You* is enough for his tremendous support and help. I feel motivated and encouraged every time I attend his meetings.

This project could never reach at this end, if **Mr. Sarabjeet Singh**, Assistant Professor, Department of CSE didn't help me in learning C/C++ and CUDA Programming. He spared time almost everyday from his busy schedule to help me. I express here my sincere gratitude for his help and support.

My sincere thanks to **Dr. T. S. Sidhu**, Director, SBS State Technical Campus, Ferozepur and to **Dr. Sanjeev Dewra**, Head, Department of ECE for the support in the form of facilities they have created and provided to students like me.

Most importantly, I would like to acknowledge the help, support, encouragement and motivation provided my parents. They have sacrificed their own needs and dreams for making me to reach at this end.

I also want to appreciate the help and support of my friends who helped me a lot in finalizing this project within the limited time frame.

I express my gratitude to all who have directly or indirectly contributed to the successful completion of this report.

Finally, I must thank GOD for giving me the environment to study, people to help, opportunities to encash and potential to succeed.

Place: SBS STC Ferozepur
Date: December 20, 2014 Durlabh Chauhan

LIST OF THESIS OUTCOMES

International Conferences Publications

1. Chauhan, Durlabh and Singh, Satvir and Singh, Sarabjeet and Banga, Vijay Kumar "Speedup of Type-1 Fuzzy Logic Systems on Graphics Processing Units Using CUDA", in Proc. International Conference on Communication, Computing and Systems, Ferozepur, Punjab, India, August, 2014, Pages 267–271.

2. Chauhan, Durlabh and Singh, Satvir and Singh, Sarabjeet and Banga, Vijay Kumar "CUDA based GPGPU to Speedup an Interval Type-2 Fuzzy Logic System", Accepted for Presentation and Publication at the IEEE Symposium Series on Computational Intelligence 2014, Orlando, Florida, December, 2014. (Withdrawn and Communicated again)

National Conferences Publications

1. Singh, Sarabjit and Singh, Satvir and Banga, Vijay Kumar and Chauhan, Durlabh "CUDA for GPGPU Applications - A Survey", in Proc. National Conference on Contemporary Techniques & Technologies in Electronics Engineering, Murthal, Sonepat, India, March, 2013, Pages 189–192.

ABSTRACT

General Purpose-computing on Graphics Processing Units (GPGPU) utilizes the tremendous computing power of Graphics Processing Unit (GPU) to accelerate Single Instruction Multiple Data (SIMD) computations by executing them in parallel in applications those are traditionally handled serially by the Central Processing Unit (CPU). GPGPU offers unprecedented application performance by offloading compute-intensive portions of the applications to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster CPU consists of a few cores optimized for sequential processing while a GPU consists of a number of (that ranges in a few tens to a few hundreds in number) smaller cores designed for handling multiple small computations, simultaneously.

Compute Unified Design Architecture (CUDA) is a parallel computing platform and programming model introduced by nVIDIA® that increases computing performance of a desktop computer substantially by harnessing the power of parallelism of the GPU. CUDA gives program developers the direct access to the virtual instruction set and memory of the parallel computational elements in CUDA enabled GPUs. The CUDA platform is accessible to software developers through CUDA-accelerated libraries, compiler directives (such as OpenACC), and extensions to industry-standard programming languages, including C, C++ and Fortran. C/C++ programmers use CUDA C/C++, compiled with *nvcc* which is nVIDIA's LLVM-based C/C++ compiler.

Artificial Intelligence (AI) is frequently applied in developing systems empowered with intellectual qualities those are characteristic of human beings, such as the ability to reason, discover meaning, generalize designs, or learn from past experiences. Problem solving, particularly in AI can be characterized as a systematic search or optimization problem through

a range of possible actions/values in order to reach some predefined goal or solution. The main subfields of AI comprises of Fuzzy Logic Systems (FLSs), Artificial Neural Networks (ANNs) and Evolutionary Algorithms (EAs).

The term *Fuzzy Logic* (FL) was introduced in the proposal of Fuzzy Set (FS) theory by Lotfi A. Zadeh in 1965. Fuzzy Logic is a computational paradigm that is based on how humans beings share/interpret information using words as linguistic variables. Fuzzy Logic handles imprecise input information much in the same way that of human brain (e.g., temperature is hot, and speed is slow, etc.) and then maps the corresponding approximate responses based on experiential knowledge.

Since introduction of FL, it has been applied in many practical engineering problems due to its capability in dealing with imprecise and inexact information. The most powerful aspect of FL is that most of human reasoning and concept formation is translated into fuzzy rules. The combination of incomplete, imprecise information and the imprecise nature of the decision-making process make fuzzy logic very effective in modeling complex engineering, business, finance and management systems which are otherwise difficult to model.

The concept of Type-2 (T2) FSs was originally introduced by Zadeh in 1975 as an extension of the concept of Type-1 (T1) FSs (ordinary Fuzzy Sets). Consequently, a T2 FLS is an extension of T1 FLS in which uncertainty among words used in fuzzy rules is represented by an additional dimension that gives more degrees of freedom for better representation of uncertainty. T2 FSs are useful in circumstances where it is difficult to determine the exact *Membership Function* (MF), i.e., Shape and Location, for the FSs due to uncertain nature of words (words means different to different people) and enable FLSs in handling a higher level of uncertainty. T2 FSs have membership grades that are themselves T1 FSs for every crisp input.

Generalized T2 FLSs are computationally intensive as it involves type-reduction that is mathematically very bulky task. Things do simplify a lot when secondary MFs are considered as interval sets, i.e., the *secondary memberships grades* are either zero or one and these FSs are termed as Interval Type-2 Fuzzy Sets or simply IT2 FSs. IT2 FSs have received the most attention as they involves much simpler mathematics than that of T2 FLSs. Therefore, IT2 FSs and FLSs has been investigated in many applications. However both kinds of FSs are being actively researched by an ever-growing number of researchers around the world as computational facilities are much better nowadays.

IT2 FSs have widely been accepted as more capable of modeling higher orders of uncertainty than that of T1 FSs. Uncertainty in IT2 FSs is represented as Footprint of Uncertainty

(FOU) using Upper Membership Functions (UMF) and Lower Membership Function (LMF). Membership grade in IT2 FSs is 1 within the range of FOU and otherwise 0.

Designing and running an FLS or an EA are true contenders for GPGPU as they involve steps those can be run in parallel due to data independency. This thesis is intended to propose a use of GPGPU for T1 and IT2 FLSs for forecasting a Mackey-Glass Time-Series to improve the runtime performance. It demonstrates the implementation and comparative runtime performances of typical T1 and IT2 FLSs on a CPU (serial) and GPU (parallel). This implementation do not use graphics API and, therefore, approach presented here is flexible, scalable, and can be used by any researcher with knowledge of C.

Two typical T1 and IT2 FLSs have been designed for forecasting of a Mackey-Glass Time-Series with varying number of inputs and fuzzy rules. FLSs with 4 inputs and 1 output is subjected to parallel computation for forecasting of Mackey-Glass Time-Series which consists of Gaussian Fuzzy sets for both T1 and IT2 FLSs. However, implication, aggregation and defuzzification methods have been kept fixed as *min* operator, *max* operator, and *height* defuzzification, respectively. Memory transfer from CPU to GPU and vice-versa is maintained as low as possible for efficient time management. In this work, the problem formulated is solved by computing number of FLS in parallel on a nVIDIA Geforce GTX 650 GPU with set of inputs varying as 128, 256, 512 and 1024. Number of fuzzy rules are also varied for all set of inputs as 10, 20, 30, ..., 100. It can be observed from the simulation results presented in this thesis that the CPU works equally fast as GPU when the system is small. As the number of fuzzy rules or the number of inputs are increased the GPU outperforms the CPU runtime. Nearly 7.83 and 12.56 times speedup have been achieved as 1024 inputs supplied in parallel to the T1 and IT2 FLS, respectively. The GPU has an initial setup overhead of kernel loading and memory transfer, however, subsequent parallel computations leads to a small increase in processing time despite a substantial increase in computational load. On the other hand, CPU has no initial cost, but computation time grows linearly with computational load much beyond GPGPU runtime.

Place: Ferozepur

Date: December 20, 2014 Durlabh Chauhan (1269835)

ABBREVIATIONS

Abbreviations	Description
AI	Artificial Intelligence
CUDA	Compute Unified Device Architecture
CPU	Central Processing Unit
FLS	Fuzzy Logic System
FS	Fuzzy Set
FOU	Footprint Of Uncertainity
UMF	Upper Membership Function
LMF	Lower Membership Function
GPU	Graphics Processing Unit
GPGPU	General Purpose computations on Graphic processing Units
T1-FLS	Type 1 Fuzzy Logic System
T2-FLS	Type 2 Fuzzy Logic System
IT2-FLS	Interval Type 2 Fuzzy Logic System
SIMD	Single Instruction Multiple Data
UOD	Universe of Discourse

NOTATIONS

Symbols	Description
$\mu(x_i)$	Clipping Level or MF Value at $x = x_i$
y_L	Defuzzified Output of Lower Bound
y_U	Defuzzified Output of Upper Bound
y	Final FLS Output
$\mu_L(x)$	Left Gaussian MF
C_i	Location of ith Singleton Fuzzy Sets
$\underline{\mu}_A(x)$	Lower Membership Function (LMF)
m	Mean of Gaussian MF
m_L	Mean of Left Gaussian MF
m_R	Mean of Right Gaussian MF
$\mu(x)$	Membership Function (MF)
$\mu_R(x)$	Right Gaussian MF
σ	Standard Deviation of Gaussian MF
M	Total Number of Fuzzy Rules
$\overline{\mu}_A(x)$	Upper Membership Function (UMF)

LIST OF FIGURES

3.1 Temperature fuzzification in Cold, Warm, and Hot fuzzy sets 12
3.2 Block Diagram for Type-2 Fuzzy Logic System 13
3.3 Interval Type-2 Fuzzy Set 14

4.1 GeForce GTX 650 16
4.2 CPU vs GPU architecture 19

5.1 CUDA Architecture 26
5.2 CUDA Process Flow 26

6.1 Interval Type-2 Fuzzy Set 31

7.1 CPU to GPU speedup ratio Vs Inputs data length (T1 FLS) 36
7.2 CPU and GPU Runtime comparison for 128 inputs (T1 FLS) 37
7.3 CPU and GPU Runtime comparison for 256 inputs (T1 FLS) 37
7.4 CPU and GPU Runtime comparison for 512 inputs (T1 FLS) 37
7.5 CPU and GPU Runtime comparison for 1024 inputs (T1 FLS) 38
7.6 CPU to GPU Speedup ratio for various input data sets (T1 FLS) 38
7.7 Average CPU to GPU Speedup ratio Vs Input data sizes for IT2 FLS 39
7.8 CPU and GPU Runtime comparison for 128 inputs (IT2 FLS) 39
7.9 CPU and GPU Runtime comparison for 256 inputs (IT2 FLS) 39
7.10 CPU and GPU Runtime comparison for 512 inputs (IT2 FLS) 40
7.11 CPU and GPU Runtime comparison for 1024 inputs (IT2 FLS) 40
7.12 CPU to GPU Speedup ratio for various input data sets (IT2 FLS) 40

LIST OF TABLES

4.1 Supported GPUs for CUDA 17

7.1 Fuzzy Rules vs CPU to GPU Speedup ratio (T1 FLS) 36
7.2 Fuzzy Rules and number of Inputs vs CPU to GPU Speedup ratio (IT2) . . . 38

CONTENTS

CERTIFICATE i

ACKNOWLEDGEMENTS v

LIST OF THESIS OUTCOMES vii

ABSTRACT viii

ABBREVIATIONS xi

NOTATIONS xii

LIST OF FIGURES xiii

LIST OF TABLES xiv

CONTENTS xv

1 INTRODUCTION 1

1.1 Introduction 1
1.2 Motivation 2
1.3 Objectives 2
1.4 Methodology 2
1.5 Thesis Outline 3

2 LITERATURE SURVEY 4

2.1 Introduction 4
2.2 Historical Developments in CUDA Investigations 4
2.3 Base Papers 7
2.3.1 Designing Fuzzy Logic Systems, 1997 8
2.3.2 Implementation of Evolutionary Fuzzy Systems, 1999 8
2.3.3 Speedup of Fuzzy Clustering Through Stream Processing on Graphics Processing Units, 2008 8

2.4 Research Gaps . . . 9
2.5 Conclusion . . . 9

3 FUZZY LOGIC SYSTEM 10
3.1 Introduction . . . 10
3.2 Fuzzy Logic . . . 10
3.3 Type-1 Fuzzy Logic System . . . 11
3.4 Type-2 Fuzzy Logic System . . . 12
3.5 Interval Type-2 Fuzzy Logic System . . . 13
3.6 Conclusion . . . 13

4 GENERAL PURPOSE COMPUTING ON GPU 15
4.1 Introduction . . . 15
4.2 GPU . . . 15
4.3 CPU and GPU Architecture . . . 19
4.4 GPU Computing . . . 19

5 COMPUTE UNIFIED DEVICE ARCHITECTURE 21
5.1 Introduction . . . 21
5.2 Compute Unified Device Architecture . . . 22
5.2.1 History . . . 22
5.2.2 CUDA Architecture . . . 23
5.3 Application Areas GPGPU . . . 23
5.4 CUDA Programming Model . . . 24
5.5 Advantages and Disadvantages . . . 27
5.6 Conclusion . . . 28

6 CUDA IMPLEMENTATION 29
6.1 Introduction . . . 29
6.2 Scope of Parallelism in Type-1 FLS . . . 29
6.3 Scope of Parallelism in Type-2 FLS . . . 31
6.4 Parallel Type-1 Fuzzy Logic System . . . 33
6.5 Parallel Interval Type-2 Fuzzy Logic System . . . 33
6.6 Conclusion . . . 34

7 RESULTS AND DISSCUSSIONS 35
7.1 Introduction . . . 35
7.2 Performance Analysis of Type-1 FLS . . . 35
7.3 Performance Analysis of Interval Type-2 FLS . . . 37
7.4 Overall Performance . . . 40
7.4.1 Performance of T1 FLS . . . 41
7.4.2 Performance of IT2 FLS . . . 41

8 CONCLUSION AND FUTURE SCOPE 42
8.1 Introduction . . . 42

8.2 T1 FLS 42
8.3 IT2 FLS 43
8.4 Future work 43
8.5 Conclusion 43

REFERENCES **45**

INDEX **48**

CHAPTER 1

INTRODUCTION

This thesis presents investigational studies on GPU computing using CUDA for improving run time of Type-1 and Type-2 Fuzzy Logic Systems. A benchmark problem of designing an FLS is developed for single stage forecasting of a Machkey-Glass Time-Series whose performance enhancement is studied when run serially on CPU and run in parallel on CPU & GPU using CUDA.

1.1 Introduction

Graphic Processing Units (GPUs) and CUDA opens up a new research domain to perform general purpose computing on hardware that is better suited for the fuzzy logic systems where scope of parallelism is huge. However, the installation of these systems on the GPUs is also difficult because many algorithms are not designed in a parallel format conducive to GPU processing. In addition, there may be too many dependencies at various stages in the algorithm that will slow down GPU processing.

Our investigations are concentrated about performance speed up of Fuzzy Logic Systems using CUDA, The term fuzzy logic was introduced with the 1965 proposal of fuzzy set theory by Lotfi A. Zadeh(Zadeh, 1965a). Fuzzy logic has been applied to many fields, from control theory to artificial intelligence.

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model created by nVIDIA and implemented to exploit the scope of GPU core to

be used for parallel computing. CUDA gives developers access to the virtual instruction set and memory of the parallel computational functions. Using CUDA, the latest nVIDIA GPUs become accessible for computation via CPU. Unlike CPUs, however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly. This approach of solving any problem in general (i.e., not exclusively graphics) on GPUs is known as GPGPU.

1.2 Motivation

FL is a well established field, however, for designing FLS increasing speedup of these systems is a relatively new approach. CUDA programming technique for GPGPU makes this topic really interesting and beneficial for the applications of evolving and running in parallel and faster. Moreover, parallel exploitation and exploration mechanisms of FLS with CUDA have a good scope to publication scope due to its nascence stage of development.

1.3 Objectives

Major objectives of the thesis are as follows:

1. To investigate Scope of parallelism in various Type-1 and Type-2 Fuzzy Logic Systems.
2. To parallelize Type-1 Fuzzy Logic System for Time Series Forecasting.
3. To parallelize Interval Type-2 Fuzzy Logic System for Time Series Forecasting.
4. To compare and analyze the results with the sequential and parallel implementation of FLSs.

1.4 Methodology

The methodology to achieve predefined thesis objectives is likely to be as follows:

1. One of the foremost requirement is to deep study and understand Fuzzy Logic System and its scope of parallelism.
2. Secondly selecting the best application preferably benchmark problem having good scope of parallelism for fulfilling our objectives.

3. Implementation of FLS in parallel and sequential and investgate the perfromances with respect to time improvement.

1.5 Thesis Outline

Chapter 2 starts with the literature survey giving an overview of GPGPU and FLS. it also presents research gaps and scope of study. Chapter 3 presents detailed introduction to FLSs and its types. Chapter 4 introduces GPUs, their architecture in general and use for parallel computations. Chapter 5 explains basic concepts of CUDA programming techniques. Chapter 6 targets implementation Type-1 and Type-2 FLS in parallel in GPU using CUDA programming model. Chapter 7 presents the simulation results of Type-1 and Type-2 FLS in serial and parallel implementation finally, conclusion and future scope is presented in Chapter 8.

CHAPTER 2

LITERATURE SURVEY

The needed detailed literature survey, to get preliminary knowledge and search scope of investigation, to design/evolve FLSs using CUDA. The principle concepts & developments that occurred till date, in these domains of research are presented in this chapter with a case built up for taking investigations that this report does.

2.1 Introduction

In this chapter, a detailed literature survey of the relevant fields is presented along with research gaps. Basics of domain related topics are introduced here and then presented the scope of further research.

2.2 Historical Developments in CUDA Investigations

CUDA has been under investigation since its introduction (June, 23, 2007). Major developments in various research domains have been described in brief as follows:

Fluid animation: Smoothed Particle Hydrodynamics (SPH) is a mesh free lagrangian particle method used for modeling flow of fluid. It was introduced by Lucy and Monaghan in 1977. SPH is a relatively new Fluid Dynamics technique to simulate motion of fluid particles. SPH is a particle based parallelizable technique, hence, more suitable for GPU

based architecture. Parallel Algorithm Design is the most important issue in any application development for CUDA. CUDA can be used for implementation of stable fluids that deals with internal and moving boundaries. The parallel implementation of CUDA is much faster as compared to sequential implementation (Nuli and Kulkarni, 2012).

Leukocyte Tracking: This application demonstrates an urgent need for dramatic speedups. The algorithm originally implemented in MATLAB, and re-implemented in C resulted in a significant performance improvement. The performance was improved further by accelerating the most computationally demanding stages using CUDA. The speedups in performance clearly demonstrate the advantages of the throughput-oriented nature of GPUs (Boyer et al., 2009).

Remote Sensing: There have been tremendous advances not only expected in optical instruments, but also in remote sensing systems.Image processing in remote sensing is particularly time consuming, and can greatly benefit from high performance computing techniques and practices to speed up processing of this type of data. With a minimum investment (hardware cost is around Rs 15000, the software used here (CUDA) is free and open source), performance gains can attain 10 to 400 times on the critical portion of the processing (Christophe et al., 2011).

Weather Forecasting: Timely weather predictions are particularly useful for severe weather events when life and property is at risk. To get weather predictions in time using latest advances in atmospheric sciences is a big challenge even on the fastest super computers. With the help of GPU, inherently parallel problems in weather forecasting can be solved effectively. GPUs have hundreds of parallel cores for execution of tens of thousands of parallel threads. So the performance in weather forecasting application is smoothened and increased with speed using GPU with CUDA (Mielikainen et al., 2012).

K-Mean Computation: K-means algorithm is one of the famous unsupervised clustering algorithms. Nowadays, desktop computers are coming equipped with programmable GPUs with plenty powerful Single Instruction Multiple Data (SIMD) processors that can support parallel data processing and high-precision computation. With the rapid advance in GPUs performance, coupled with recent improvements in their programmability, made it possible to parallelize K-means with personal computers. In this algorithm, both data objects assignment and k centroids recalculation of traditional k-means are parallel performed on the GPU (Hong-Tao et al., 2009).

Bayesian Computation: The package cudaBayesreg is used to implement a Bayesian multilevel model for the analysis of brain functional magnetic resonance imaging (fMRI)

data in the CUDA environment. The CUDA implementation of the Bayesian model has been able to reduce significantly the runtime processing of the MCMC simulations. The increased performance comes from the use of separate threads for fitting the linear regression model at each voxel in parallel (da Silva, 2010).

Molecular Dynamics Simulation: Molecular Dynamics is a computationally intensive method for studying the natural time-evolution of a system of atoms using Newtons classical equations of motion. MD has always been limited more by the current available computing power. Researchers in this field have typically focused their efforts to simplify models and identify what can be neglected to still obtain acceptable results. HPC on GPU is the key in making biologically relevant calculations tractable without compromise (Liu et al., 2008).

Continuous Space Language Model: The Continuous Space Language Model (CSLM), introduced by Schwenk along with an open source implementation, provides an alternative to the n-gram back off model and allows true interpolation of the probabilities of unseen n-grams. The CSLM algorithm is highly computationally intensive and is a good candidate for implementation with CUDA. CUBLAS is a CUDA implementation of BLAS (Basic Linear Algebra Subprogram), which performs matrix multiplication operations (Thompson and Anderson, 2012).

Vehicle Routing: Many of identical vehicles with a given capacity are located at a central depot. They need to service a set of customer orders. Each customer orders from a specific location and has specific size. Travel costs between different locations are available. The goal is to design a least-cost set of routes for all the vehicles so that all customers are visited once and vehicle capacities are adhered to. Our main goal is to carefully asses how well local search implementation can fit the given hardware, identify limiting factors and to resolve them (Schulz, 2013).

Swarm Based Algorithms: Artificial Intelligence (AI) is one of the most dominating paradigms of modern research and inculcates computationally intensive algorithms. GPU is being investigated to increase performance of these AI algorithms and brief survey is given below:

1. **Ant Colony Optimization**: ACO parallel implementations can be divided into two general approaches. The first one is the parallel execution of the ants construction phase in a single colony. It aims to accelerate computations by distributing ants to computing elements. The second that was introduced by Sttzle is the execution of multiple ant colonies. The ACO metaheuristic, the MaxMin Ant System (MMAS)

algorithm and its application to the Traveling Salesman Problem (TSP). The selection of MMAS and TSP is to focus on algorithmic aspects of ACO and technical aspects of GPU computing that are not problem dependent and to compare with the work of Sttzle and Hoos (Delévacq et al., 2013).

2. **Particle Swarm Optimization**: Advanced Driving Assistance Systems (ADAS) needs road sign detection. A novel approach based on both sign shape and color which uses Particle Swarm Optimization (PSO) for detection can be used. A single fitness function may be used both to detect a sign belonging to a certain category and, in parallel, to estimate its actual position with respect to the camera reference frame. To speed up execution time, the algorithm exploits the parallelism available with GPUs these days (Mussi et al., 2009).

3. **Genetic Algorithms**: Genetic Algorithms (GA) is candidate for parallel computing since population members fitness can be evaluated in parallel. GA uses a number of other operations which, if performed in parallel, can lead to faster evolutionary performance. The binary and real-coded genetic algorithms using CUDA may be executed in parallel. Significant speed-ups results over the sequential GA (Arora et al., 2010).

All applications of GPGPU with CUDA results in tremendous speedup in performance. Leukocyte Tracking implementation achieved an overall speedup of 199.9x using a desktop system with an NVIDIA GeForce GTX 280 GPU, as compared to a speedup of 7.6x on the fastest available multicore CPU system. The new Molecular Dynamics algorithm using CUDA leads to a performance improvement. Weather forecasting resulted in a reduction in processing time from 16928ms on CPU to 43.5ms on a Graphics Processing Unit (GPU). In Remote Sensing performance gains can attain 10 to 400 times. The ACO shows speedups of up to 23.60 with solution quality similar to the original sequential implementation.

2.3 Base Papers

As usual this project is not output/extention of a single research paper. Work presented in three different papers is collaborated here to formulate out research objectives. These are discussed as follows:

2.3.1 Designing Fuzzy Logic Systems, 1997

Authors: Jerry M. Mendel, George C. Mouzoris (IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing): In this, a formulation of a fuzzy logic system (FLS) that can be used to construct nonparametric models of nonlinear processes, given only input-output data was presented. In order to effectively construct such models, several design methods with different properties and features were discussed.This compares and illustrate systems designed with each one of the methods, using an example on the predictive modeling of a nonlinear dynamic (chaotic) system (Mendel and Mouzouris, 1997).

2.3.2 Implementation of Evolutionary Fuzzy Systems, 1999

Authors: Yuhui Shi,Russel Eberhart, Yaobin Chen (IEEE Transactions on Fuzzy Systems): Evolutionary fuzzy systems are discussed in which the membership function shapes and types and the fuzzy rule set including the number of rules inside it are evolved using a genetic (evolutionary) algorithm. In addition, the genetic parameters (operators) of the evolutionary algorithm are adapted via a fuzzy system. Benefits of the methodology are illustrated in the process of classifying the iris data set. Possible extensions of the methods are summarized (Shi et al., 1999).

2.3.3 Speedup of Fuzzy Clustering Through Stream Processing on Graphics Processing Units, 2008

Authors: Derek T. Anderson, Robert H. Luke, and James M. Keller (IEEE Transactions on Fuzzy Systems): As the number of data points, feature dimensionality, and number of centers for clustering algorithms increase, computational tractability becomes a problem. The fuzzy c-means has a large degree of inherent algorithmic parallelism that modern CPU architectures do not exploit. Many pattern recognition algorithms can be sped up on a graphics processing unit (GPU) as long as the majority of computation at various stages and the components are not dependent on each other. This paper presents a generalized method for offloading fuzzy clustering to a GPU, while maintaining control over the number of data points, feature dimensionality, and the number of cluster centers. GPU-based clustering is a high-performance low-cost solution that frees up the CPU. Our results show a speed increase of over two orders of magnitude for particular clustering configurations and platforms (Anderson et al., 2008).

2.4 Research Gaps

Lack of journal publications on accelerating the FLSs using CUDA, makes it clear for us to work on this field as it will be an open ended research and the objectives we proposed in the introduction are new and open ended for further investigations. As till date no work is published in parallel computing of generalized T2 FLS. Even after 4 years of the introduction of nVIDIA's CUDA platform for parallel computing, lot of research domains in FLSs are still not made parallel in all possible ways even after good scope of parallelism.

2.5 Conclusion

After doing the literature survey in the required domains it is clear that the topic is new for research with open ended results. Scope of parallelism is there in many FLS whether IT2 and T2 FLS. FLS are not implemented with GPGPU using CUDA in order to achieve a degree of performance in application runtime which encourage us to work more hard to achieve the proposed objectives.

CHAPTER 3

FUZZY LOGIC SYSTEM

In this chapter, a brief explanation of Fuzzy Logic System is discussed describing the terms and procedures linked with it. It also presents introduction about the Type-1 and Type-2 and Interval Type-2 Fuzzy Logic Systems.

3.1 Introduction

Fuzzy logic is a form of many-valued logic; it deals with reasoning that is approximate rather than fixed and exact. Compared to traditional binary sets (where variables may take on true or false values) fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false. Furthermore, when linguistic variables are used, these degrees may be managed by specific functions. Irrationality can be described in terms of what is known as the fuzzjective.

3.2 Fuzzy Logic

The term *Fuzzy Logic* was introduced with the 1965 proposal of fuzzy set theory by Lotfi A. Zadeh (Zadeh, 1965a). Fuzzy logic has been applied to many fields, from control theory to artificial intelligence. Fuzzy logics however had been studied since the 1920s as infinite-valued logics notably by ukasiewicz and Tarski (Mundici and Olivetti, 1998).

Classical logic only permits propositions having a value of truth or falsity. The notion of whether $1 + 1 = 2$ is absolute, immutable, mathematical truth. However, there exist certain propositions with variable answers, such as asking various people to identify a color. The notion of truth doesn't fall by the wayside, but rather a means of representing and reasoning over partial knowledge is afforded, by aggregating all possible outcomes into a dimensional spectrum.

Both degrees of truth and probabilities range between 0 and 1 and hence may seem similar at first. For example, let a 100ml glass contain 30ml of water. Then we may consider two concepts: Empty and Full. The meaning of each of them can be represented by a certain fuzzy set. Then one might define the glass as being 0.7 empty and 0.3 full. Note that the concept of emptiness would be subjective and thus would depend on the observer or designer. Another designer might equally well design a set membership function where the glass would be considered full for all values down to 50ml. It is essential to realize that fuzzy logic uses truth degrees as a mathematical model of the vagueness phenomenon while probability is a mathematical model of ignorance.

A basic application might characterize subranges of a continuous variable. For instance, a temperature measurement for anti-lock brakes might have several separate membership functions defining particular temperature ranges needed to control the brakes properly. Each function maps the same temperature value to a truth value in the 0 to 1 range. These truth values can then be used to determine how the brakes should be controlled.

In this image, the meanings of the expressions cold, warm, and hot are represented by functions mapping a temperature scale. A point on that scale has three "truth values"one for each of the three functions. The vertical line in the image represents a particular temperature that the three arrows (truth values) gauge. Since the red arrow points to zero, this temperature may be interpreted as "not hot". The orange arrow (pointing at 0.2) may describe it as "slightly warm" and the blue arrow (pointing at 0.8) "fairly cold".

3.3 Type-1 Fuzzy Logic System

Theory of FLS given by Zadeh a fuzzy set is defined for a particular domain, and it is characterized by a membership function that maps elements from the domain to a real valued numbers (Zadeh, 1965b, 1975). Mendel and many researchers gave numerous methods to design and implement FLS for various applications (Karnik and Mendel, 1999; Mendel, 1995, 2000; Mouzouris and Mendel, 1997). Theory of FLS given by Zadeh a fuzzy set is

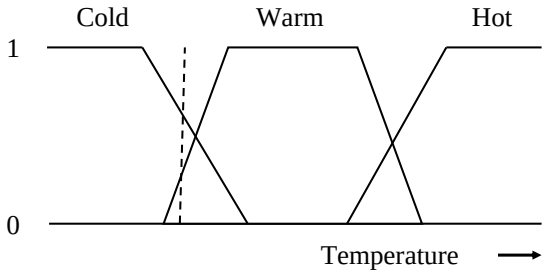

FIGURE 3.1: Temperature fuzzification in Cold, Warm, and Hot fuzzy sets

defined for a particular domain, and it is characterized by a membership function that maps elements from the domain to a real valued numbers. The typical approach to representing T1 fuzzy sets in a computer system is to discretise the sets by maintaining an array of pairs of domain values and their associated membership grades. This inevitably leads to inaccuracies and speed problems in large systems. It is clear from the review of journals that a T1 FLS is unable to handle uncertainties because of its really simple methodological algebra.

3.4 Type-2 Fuzzy Logic System

The concept of T2 FSs was originally introduced by Zadeh as an extension of the concept of T1 FSs in 1975 (Zadeh, 1975). T2 FLS are an extension of T1 FLS in which uncertainty is represented by an additional dimension (Liang and Mendel, 2000). This ancillary third dimension in T2 FLS gives more degrees of freedom for better representation of uncertainty compared to T1 fuzzy sets. T2 FSs are useful in circumstances where it is difficult to determine the exact membership function for a FS. Using T2 FLS provides the capability of handling a higher level of uncertainty and provides a number of missing components that have held back successful deployment of FS in human decision making.

A T2 FLS includes fuzzifier, rule base, fuzzy inference engine, and output processor as shown in Fig. 3.2. The output processor includes type-reducer and defuzzifier which generates a T1 FS output (from the type-reducer) or a crisp number (from the defuzzifier). A T2 FLS is characterized using T2 FSs for antecedents and/or consequents and IF-THEN rules.

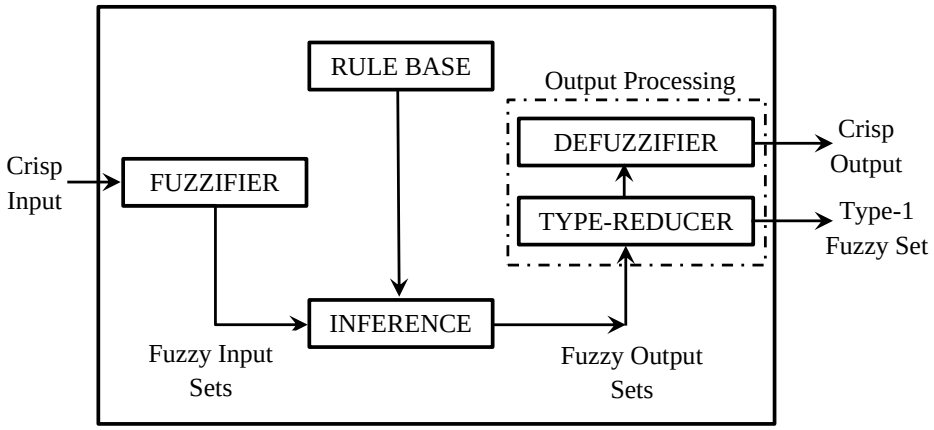

FIGURE 3.2: Block Diagram for Type-2 Fuzzy Logic System

3.5 Interval Type-2 Fuzzy Logic System

Generalized T2 FLSs are computationally more intensive as compared to T1 FLS as former includes FSs those are 3-dimensional in nature. Things do simplify when secondary membership functions are considered as interval sets, i.e., the secondary membership values are either 0 or 1 and set are referred as IT2 FSs or simply IT2 FSs. IT2 FSs have received the most investigational interests as they involve mathematics that is simpler than that of generalized T2 FSs. Therefore, literature available about IT2 FSs is more as compared to that of generalized T2 FSs. Now a days, both kinds of fuzzy sets are being actively investigated by an ever-growing number of researchers around the world.

IT2 FSs have widely been accepted as they provide more freedom degree in modeling higher orders of uncertainty than T1 FSs. This property has been the driving force behind more of the advancements in theories and applications of IT2 FSs and FLSs. IT2 FSs are represented by upper and lower bounds of uncertainty called Upper Membership Function (UMF) and Lower Membership Function (LMF) as shown in Fig. 3.3. The region between upper and lower bounds of Uncertainty is termed as Footprint of uncertainty (FOU).

3.6 Conclusion

FLSs are well established problem solving mathematica and has been used widely for various applications. The types of FLS discussed above have their own advantages and limitations. T1-FLS is easy to solve and implement but is not accurate for handling any sort of uncertainty. On the other hand T2-FLS are able to handle uncertainties up to a great extent but

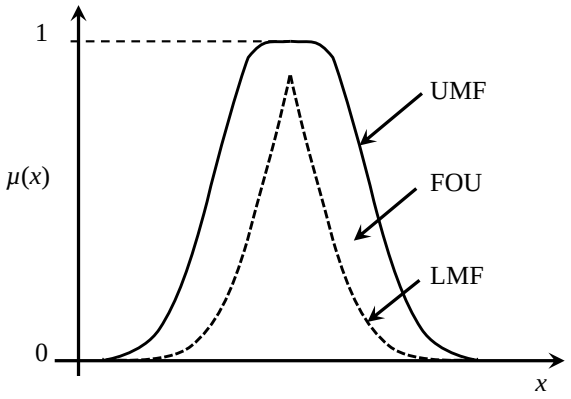

FIGURE 3.3: Interval Type-2 Fuzzy Set

is really very computationally intensive to implement. IT2-FLS is a unique method to solve T2-FLS with comparative less computations and easy to implement.

CHAPTER 4

GENERAL PURPOSE COMPUTING ON GPU

In this chapter, the GPGPU is unfolded to explore its meaning and relevance with research areas and most importantly the device architecture is discussed.

4.1 Introduction

The term GPGPU was coined and was founded by Mark Harris in 2002. GPGPU stands for General-Purpose computation on Graphics Processing Units. Graphics Processing Units (GPUs) are high-performance many-core processors that can be used to accelerate a wide range of applications.The reason for the GPUs advantage compared to the CPU is the parallel processing hierarchy of GPU's with SIMD (Single Instruction Multiple Data) accessing feature.

4.2 GPU

GPU design-idea has been to sacrifice the complex, low serial latency architecture of the desktop CPU and instead focusing on a simple, highly parallel, high bandwidth architecture instead, as this is more important in both gaming and graphic intense applications,the main

areas of use for GPUs today. Not only is there more potential performance and performance/Watt in a GPU, but also, the GPUs have an additional advantage. Compared to a graphics memory, CPU memory tends to be much more bandwidth constrained, thus it is comparatively more difficult to extract all the theoretical FLOPS from the processor. This is one of the principal reasons that performance on data-intensive applications almost never scales linearly on multicore CPUs. GPU architectures, on the other hand, have always been designed as data throughput processors, so the FLOPS to bandwidth ratio is much more favourable.

GPUs have been known to users since quite a long time as a graphics rendering coprocessor to the host PC, to render cool graphic effects in multimedia based applications such as gaming, animation etc. But now the technology inside the GPU has crossed that limit and being used also for many computing applications other than rendering graphics.

The research community has clearly demonstrated how non-graphics-related computing can be performed on the GPUs, with more than a thousands of papers published so far in this field. So GPGPU is use of GPU computing for general purpose applications.

GPU is typically a computer card, installed into a PCI Express 16x slot. Market leaders those manufacture such graphics cards are nVIDIA, Intel, and AMD (ATI), etc.

FIGURE 4.1: GeForce GTX 650

TABLE 4.1: Supported GPUs for CUDA

Compute Capability	**Architecture**	**GPUs**	**GPUs Cards**
1.0	Tesla	G80, G92, G92b, G94, G94b	GeForce GT 420*, GeForce 8800 Ultra, GeForce 8800 GTX, GeForce GT 340*, GeForce GT 330*, GeForce GT 320*, GeForce 315*, GeForce 310*, GeForce 9800 GT, GeForce 9600 GT, GeForce 9400GT, Quadro FX 5600, Quadro FX 4600, Quadro Plex 2100 S4, Tesla C870, Tesla D870, Tesla S870
1.1	Tesla	G86, G84, G98, G96, G96b, G94, G94b, G92, G92b	GeForce G110M, GeForce 9300M GS, GeForce 9200M GS, GeForce 9100M G, GeForce 8400M GT, GeForce 8600 GT, GeForce 8600 GTS, GeForce G105M, Quadro FX 4700 X2, Quadro FX 3700, Quadro FX 1800, Quadro FX 1700, Quadro FX 580, Quadro FX 570, Quadro FX 470, Quadro FX 380, Quadro FX 370, Quadro FX 370 Low Profile, Quadro NVS 450, Quadro NVS 420, Quadro NVS 290, Quadro NVS 295, Quadro Plex 2100 D4, Quadro FX 3800M, Quadro FX 3700M, Quadro FX 3600M, Quadro FX 2800M, Quadro FX 2700M, Quadro FX 1700M, Quadro FX 1600M, Quadro FX 770M, Quadro FX 570M, Quadro FX 370M, Quadro FX 360M, Quadro NVS 320M, Quadro NVS 160M, Quadro NVS 150M, Quadro NVS 140M, Quadro NVS 135M, Quadro NVS 130M, Quadro NVS 450, Quadro NVS 420, Quadro NVS 295
1.2	Tesla	GT218, GT216, GT215	GeForce GT 240, GeForce GT 220*, GeForce 210*, GeForce GTS 360M, GeForce GTS 350M, GeForce GT 335M, GeForce GT 330M, GeForce GT 325M, GeForce GT 240M, GeForce G210M, GeForce 310M, GeForce 305M, Quadro FX 380 Low Profile, NVIDIA NVS 300, Quadro FX 1800M, Quadro FX 880M, Quadro FX 380M, NVIDIA NVS 300, NVS 5100M, NVS 3100M, NVS 2100M, ION
1.3	Tesla	GT200, GT200b	GeForce GTX 280, GeForce GTX 275, GeForce GTX 260, Quadro FX 5800, Quadro FX 4800, Quadro FX 4800 for Mac, Quadro FX 3800, Quadro CX, Quadro Plex 2200 D2, Tesla C1060, Tesla S1070, Tesla M1060
2.0	Fermi	GF100, GF110	GeForce GTX 590, GeForce GTX 580, GeForce GTX 570, GeForce GTX 480, GeForce GTX 470, GeForce GTX 465, GeForce GTX 480M, Quadro 6000, Quadro 5000, Quadro 4000, Quadro 4000 for Mac, Quadro Plex 7000, Quadro 5010M, Quadro 5000M, Tesla C2075, Tesla C2050/C2070, Tesla M2050/ M2070/ M2075/ M2090

Continued on next page

Table 4.1 – continued from previous page

Compute Capability	**Architecture**	**GPUs**	**GPUs Cards**
2.1	Fermi	GF104, GF106, GF108, CGF114, GF116, GF119	GeForce GTX 550 Ti, GeForce GTX 460, GeForce GTS 450, GeForce GTS 450*, GeForce GT 640 (GDDR3), GeForce GT 630, GeForce GT 620, GeForce GT 610, GeForce GT 520, GeForce GT 440, GeForce GT 440*, GeForce GT 430, GeForce GT 430*, GeForce GTX 675M, GeForce GTX 670M, GeForce GT 635M, GeForce GT 630M, GeForce GT 625M, GeForce GT 720M, GeForce GT 620M, GeForce 710M, GeForce 610M, GeForce GTX 580M, GeForce GTX 570M, GeForce GTX 560M, GeForce GT 555M, GeForce GT 550M, GeForce GT 540M, GeForce GT 525M, GeForce GT 520MX, GeForce GT 520M, GeForce GTX 485M, GeForce GTX 470M, GeForce GTX 460M, GeForce GT 445M, GeForce GT 435M, GeForce GT 420M, GeForce GT 415M, GeForce 710M, GeForce 410M, Quadro 2000, Quadro 2000D, Quadro 600, Quadro 410, Quadro 4000M, Quadro 3000M, Quadro 2000M, Quadro 1000M, NVS 5400M, NVS 5200M, NVS 4200M
3.0	Kepler	GK104, GK106, GK107	GeForce GTX 770, GeForce GTX 760, GeForce GT 740, GeForce GTX 690, GeForce GTX 680, GeForce GTX 670, GeForce GTX 660 Ti, GeForce GTX 660, GeForce GTX 650 Ti BOOST, GeForce GTX 650 Ti, GeForce GTX 650, GeForce GTX 780M, GeForce GTX 770M, GeForce GTX 765M, GeForce GTX 760M, GeForce GTX 680MX, GeForce GTX 680M, GeForce GTX 675MX, GeForce GTX 670MX, GeForce GTX 660M, GeForce GT 750M, GeForce GT 650M, GeForce GT 745M, GeForce GT 645M, GeForce GT 740M, GeForce GT 730M, GeForce GT 640M, GeForce GT 640M LE, GeForce GT 735M, GeForce GT 730M, Quadro K5000, Quadro K4200, Quadro K4000, Quadro K2000, Quadro K2000D, Quadro K600, Quadro K420, Quadro K500M, Quadro K510M, Quadro K610M, Quadro K1000M, Quadro K2000M, Quadro K1100M, Quadro K2100M, Quadro K3000M, Quadro K3100M, Quadro K4000M, Quadro K5000M, Quadro K4100M, Quadro K5100M, Tesla K10
3.5	Kepler	GK110, GK208	GeForce GTX TITAN Z, GeForce GTX TITAN Black, GeForce GTX TITAN, GeForce GTX 780 Ti, GeForce GTX 780, GeForce GT 640 (GDDR5), GeForce GT 630 v2, Quadro K6000, Quadro K5200, Tesla K40, Tesla K20x, Tesla K20
5.0	Maxwell	GM107, GM108	GeForce GTX 750 Ti, GeForce GTX 750, GeForce GTX 860M, GeForce GTX 850M, GeForce 845M, GeForce 840M, GeForce 830M, Quadro K2200, Quadro K620
5.2	Maxwell	GM204	GeForce GTX 980, GeForce GTX 970, GeForce GTX 980M, GeForce GTX 970M

4.3 CPU and GPU Architecture

Basic difference between CPU and GPU is depicted in the Fig. 4.2

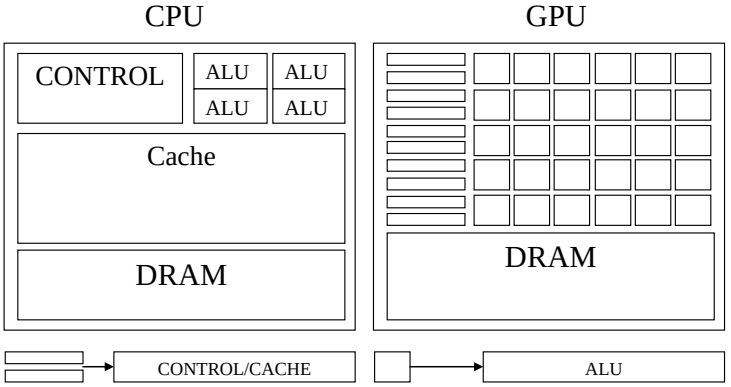

FIGURE 4.2: CPU vs GPU architecture

4.4 GPU Computing

GPU computing is the use of a GPU together with a CPU to accelerate general-purpose scientific and engineering applications.CPU controls all the computations.CPU sends tasks and data to GPU, GPU performs computations on data and sends back results to CPU.In this way, in context of GPU computing GPU is called as DEVICE and CPU is called as HOST.

GPU computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster. GPU accelerates critical paths of application, i.e., we can target to remove bottlenecks of execution speed in our code.

GPUs consist of thousands of smaller, more efficient cores designed for parallel performance. CPUs consist of a few cores optimized for serial processing, e.g., Intel Pentium Dual Core processors have 2 cores, Quad core have 4, which are very few. Serial portions of the code run on the CPU while parallel portions run on the GPU.

The GPU devotes more transistors for computation, i.e., GPUs contain much larger number of dedicated ALUs then CPUs. Each processing unit on GPU contains local memory that improves data manipulation and reduces fetch time.

CHAPTER 5

COMPUTE UNIFIED DEVICE ARCHITECTURE

In this chapter CUDA which is typically a programming model is explored to users knowledge and its advantages and limitations are also well discussed. CUDA has variety of applications too which will be covered in this chapter.

5.1 Introduction

CUDA is a parallel computing platform and programming model created by nVIDIA and implemented by the GPUs that they produce. CUDA gives developers access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the latest nvidia GPUs become accessible for computation like CPUs. Unlike CPUs, however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly. This approach of solving general-purpose (i.e., not exclusively graphics) problems on GPUs is known as GPGPU.

5.2 Compute Unified Device Architecture

The CUDA platform is accessible to software developers through CUDA-accelerated libraries, compiler directives (such as OpenACC), and extensions to industry-standard programming languages, including *C*, *C++* and *Fortran*. C/C++ programmers use *CUDA C/C++*, compiled with *nvcc*, nVIDIA's *LLVM*-based *C/C++* compiler, and Fortran programmers can use *CUDA Fortran*, compiled with the PGI CUDA Fortran compiler from The Portland Group.

In addition to libraries, compiler directives, *CUDA C/C++* and *CUDA Fortran*, the CUDA platform supports other computational interfaces, including the Khronos Group's *OpenCL*, Microsoft's *Direct Compute*, and *C++ AMP*. Third party wrappers are also available for *Python*, *Perl*, *Fortran*, *Java*, *Ruby*, *Lua*, *Haskell*, *MATLAB*, *IDL*, and native support in Mathematica.

In the computer game industry, GPUs are used not only for graphics rendering but also in game physics calculations (physical effects like debris, smoke, fire, fluids); examples include PhysX and Bullet. CUDA has also been used to accelerate non-graphical applications in computational biology, cryptography and other fields by an order of magnitude or more.

CUDA provides both a low level API and a higher level API. The initial CUDA SDK was made public on 15 February 2007, for Microsoft Windows and Linux. Mac OS X support was later added in version 2.0 which supersedes the beta released February 14, 2008. CUDA works with all nVIDIA's GPUs from the G8x series onwards, including GeForce, Quadro and the Tesla line. CUDA is compatible with most standard operating systems. nVIDIA states that programs developed for the G8x series will also work without modification on all future nVIDIA video cards, due to binary compatibility.

5.2.1 History

The first GPUs were designed as graphics accelerators, supporting only specific fixed-function pipelines. Starting in the late 1990s, the hardware became increasingly programmable, culminating in nVIDIA's first GPU in 1999. Less than a year after nVIDIA coined the term GPU, artists and game developers weren't the only ones doing ground-breaking work with the technology: Researchers were tapping its excellent floating point performance. The General Purpose GPU (GPGPU) movement had dawned.

But GPGPU was far from easy back then, even for those who knew graphics programming languages such as OpenGL. Developers had to map scientific calculations onto problems

that could be represented by triangles and polygons. GPGPU was practically off-limits to those who hadn't memorized the latest graphics APIs until a group of Stanford University researchers set out to reimagine the GPU as a "streaming processor."

In 2003, a team of researchers led by Ian Buck unveiled Brook, the first widely adopted programming model to extend C with data-parallel constructs. Using concepts such as streams, kernels and reduction operators, the Brook compiler and runtime system exposed the GPU as a general-purpose processor in a high-level language. Most importantly, Brook programs were not only easier to write than hand-tuned GPU code, they were seven times faster than similar existing code.

nVIDIA knew that blazingly fast hardware had to be coupled with intuitive software and hardware tools, and invited Ian Buck to join the company and start evolving a solution to seamlessly run C on the GPU. Putting the software and hardware together, nVIDIA unveiled CUDA in 2006, the world's first solution for general-computing on GPUs.

5.2.2 CUDA Architecture

There are two main parts of the CUDA architecture:

1. **Host (CPU)**: It carries Single Program, Single Data (SISD) programming model.
2. **Device (GPU)**: It carries Single Program, Multiple Data (SIMD) programming model.

In the Device (GPU) a Grid is defined as the group of threads all running the same kernel. a user can run multiple grids at once. Grids are composed of blocks Each block is a logical unit containing a number of coordinating threads and some amount of shared memory.

5.3 Application Areas GPGPU

Major application domains of GPGPU reported till date are outlined here:

1. Research: Higher Education and Super computing
   (a) Computational Chemistry And Biology
   (b) Numerical Analytics
   (c) Physics

2. Defense and Intelligence
3. Computational Finance
4. Manufacturing: CAD and CAE
   (a) Computational Fluid Dynamics
   (b) Computational Structural Mechanics
   (c) Computer Aided Design
   (d) Electronic Design Automation
5. Media and Entertainment
   (a) Animation
   (b) Modeling and Rendering
   (c) Color Correction and Grain Management
   (d) Compositing
   (e) Finishing and effects editing
   (f) Encoding and Digital Distribution
   (g) On-Air Graphics
   (h) On-Set
   (i) Review and Stereo Tools
   (j) Simulation
   (k) Weather Graphics
6. Oil and Gas

Accelerated computing has revolutionized the HPC industry. There are over two hundred applications across a wide range of fields already optimized for GPUs to help you accelerate your work.

5.4 CUDA Programming Model

CUDA is a parallel computing platform and programming model introduced by nVIDIA that increases computing performance substantially by harnessing the power of parallelism of the GPU. CUDA gives program developers the direct access to the virtual instruction set

and memory of the parallel computational elements in CUDA enabled GPUs. The CUDA platform is accessible to software developers through CUDA-accelerated libraries, compiler directives (such as OpenACC), and extensions to industry-standard programming languages, including C, C++ and Fortran. C/C++ programmers use CUDA C/C++, compiled with *nvcc* which is nVIDIA's LLVM-based C/C++ compiler. A C/C++ program using CUDA can interface with one GPU or multiple GPUs and can be identified and utilized in parallel, allowing for unprecedented processing power on a desktop computers.

CUDA allows multiple kernels to be run simultaneously on GPU cores. CUDA refers to each kernel as a grid. A grid is a collection of blocks. Each block runs the same kernel, however, is independent of each other (this has significance in terms of access to memory types). A block contains threads, which are the smallest divisible unit on a GPU.

A thread block is a number of SIMD threads that work on core at a given time. Threads can exchange information through the shared memory and can be synchronized. The operations are systematized as a grid of thread blocks. For parallel operation the programming model allows a developer to partition a program into several subprograms, each of which is executed independently on a block. Each subprogram can be further divided into finer pieces that perform the same function but execute on different threads independently within the same block. For data set parallelism, data sets can be divided in to smaller chunks that are stored in the shared memory, and each chunk is visible to all threads of the same block. This local data arrangement approach reduces the need to access off-chip global memory, which reduces data access time.

The next critical component of a CUDA application is the memory model. There are multiple types of memory and each has different access times. The GPU is broken up into read-write per thread registers, read-write per thread local memory, read-write per-block shared memory, read-write per-grid global memory, read-only per-grid constant memory, and read-only per-grid texture memory.

Texture and constant memory have relatively small access latency times, while global memory has the largest access latency time. Applications should minimize the number of global memory reads and writes. This is typically achieved by having each thread read its data from global memory and store its content into shared memory.

The basic structure of a CUDA code comprises of allocation of memory space (using **cudaMalloc** function) on device (GPU) and (using regular **malloc** function) on host (CPU). Data which is copied from the host to the device for the call of kernel routine to be executed on the GPU (using function **cudaMemcpy**) also defines the number of threads and their

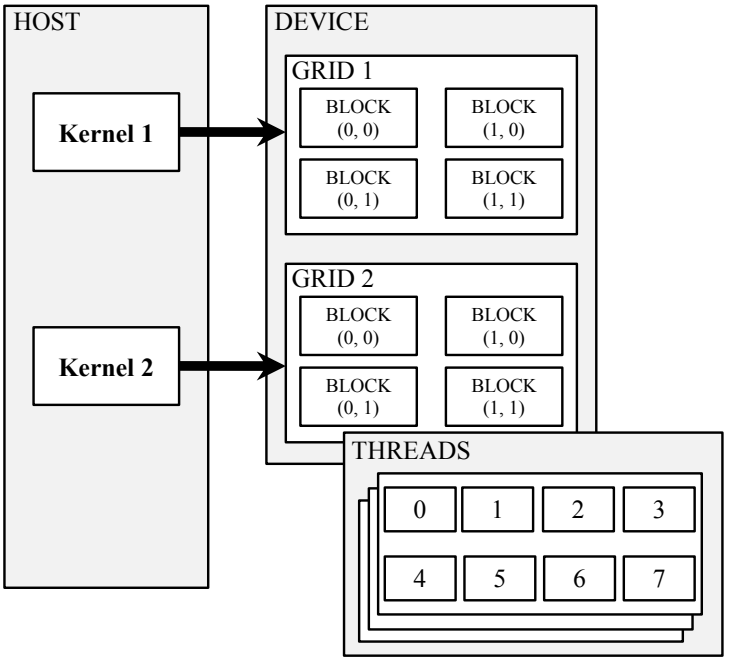

FIGURE 5.1: CUDA Architecture

physical structure. Kernel is prefixed with the **__global__** keyword. Results are transferred from GPU to CPU in the same fashion as data is copied from host to device.

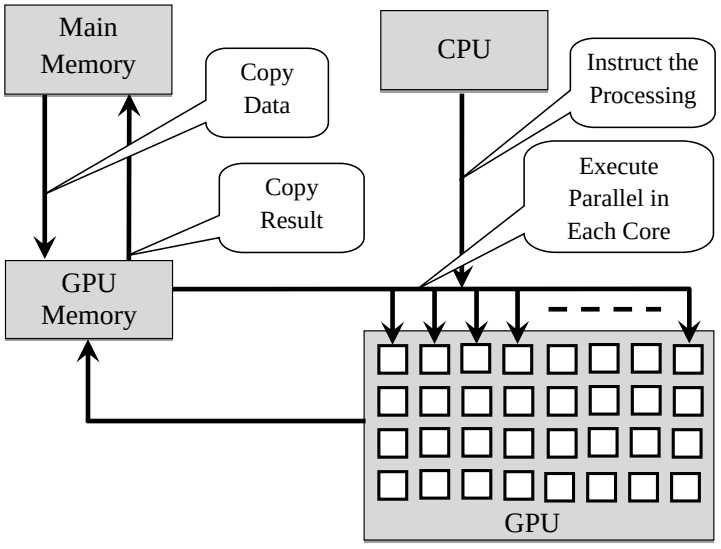

FIGURE 5.2: CUDA Process Flow

5.5 Advantages and Disadvantages

CUDA has various advantages than traditional GPU computing using graphics API. Some of them are discussed below in brief:

1. CUDA has capability of Scattered reads, i.e., code can read from arbitrary addresses in memory.
2. Availability Unified virtual memory (CUDA 4.0 and above).
3. Unified memory (CUDA 6.0 and above) for users with fast interface.
4. CUDA exposes a fast shared memory region (up to 48 KB per multi-processor) that can be shared amongst threads. This can be used as a user-managed cache, enabling higher bandwidth than is possible using texture lookups.
5. Faster downloads and readbacks to and from the GPU
6. Last but not the least it offers full support for integer and bitwise operations, including integer texture lookups.

Unfortunately CUDA do carry some limitations with it. some of them are discussed below in brief:

1. CUDA does not support the full C standard, as it runs host code through a C++ compiler, which makes some valid C (but invalid C++) code fail to compile.
2. Interoperability with rendering languages such as OpenGL is one-way, with access to OpenGL having access to registered CUDA memory but CUDA not having access to OpenGL memory.
3. Copying between host and device memory may incur a performance hit due to system bus bandwidth and latency (this can be partly alleviated with asynchronous memory transfers, handled by the GPU's DMA engine)
4. Threads should be running in groups of at least 32 for best performance, with total number of threads numbering in the thousands. Branches in the program code do not affect performance significantly, provided that each of 32 threads takes the same execution path; the SIMD execution model becomes a significant limitation for any inherently divergent task (e.g. traversing a space partitioning data structure during ray tracing).

5. Unlike OpenCL, CUDA-enabled GPUs are only available from Nvidia
6. No emulator or fallback functionality is available for modern revisions
7. Valid C/C++ may sometimes be flagged and prevent compilation due to optimization techniques the compiler is required to employ to use limited resources.

5.6 Conclusion

CUDA is not only a programming model to program a GPU but its a technical language to efficiently parallelize codes running on CPU. CUDA is powerful programming platform for increasing the computing performance of the applications and codes running and made for CPU's classically. It has advantages over CPU and do have some limitations too but as far as computing performance is concerned it is the right player for it.

CHAPTER 6

CUDA IMPLEMENTATION

In this chapter, serial and parallel implementation of a fuzzy logic system is introduced in which we will identify the scope of parallelism in the modules which are implemented so far.

6.1 Introduction

FLS are itself fuzzy so the implementation part is the difficult one. One need to know the easiest mathematics behind the code to generalize the system and to evolve it with a unique technique to explore and exploit the major areas with high scope of parallelism. T1 and IT2 FLS are needed to explore for possible scope of parallelism in the serial implementation. The set of mathematical equation used in implementation decides the efficient way to exploit the parallelism in the code.

6.2 Scope of Parallelism in Type-1 FLS

Theory of FLS given by Zadeh a fuzzy set is defined for a particular domain, and it is characterized by a membership function that maps elements from the domain to a real valued numbers (Zadeh, 1965b, 1975). Mendel and many researchers gave numerous methods to design and implement FLS for various applications (Karnik and Mendel, 1999; Mendel,

1995, 2000; Mouzouris and Mendel, 1997). Theory of FLS given by Zadeh a fuzzy set is defined for a particular domain, and it is characterized by a membership function that maps elements from the domain to a real valued numbers.

In this work every input has been fuzzified using four arbitrary located Gaussian membership functions that are characterized by two parameters, i.e., mean (m) and standard deviation (σ). Gaussian membership function is symmetrical about its mean and is expressed mathematically as (6.1)

$$\mu(x) = \exp\left(\frac{-(x-m)^2}{2\sigma^2}\right) \tag{6.1}$$

In this work, an FLS with 4 inputs and 1 output is subjected to parallel computation for forecasting of Mackey-Glass time series (Karnik and Mendel, 1999). For implication and aggregation, *min* and *max* operator are investigated. The defuzzification is performed with the height defuzzification method as symmetrical shaped gaussian fuzzy sets have been used to fuzzify consequent. Output of the height defuzzifier is given by (6.2)

$$y = \frac{\sum_{i=1}^{M} C_i \mu(x_i)}{\sum_{i=1}^{M} \mu(x_i)} \tag{6.2}$$

Here C represents the location of singleton consequents fuzzy sets and $\mu(x_i)$ represent clipping level for each rule after implication and M represents total number of fuzzy rules. However, for implementation of the FLS defined above in CUDA, the foremost and the most crucial step is to allocate the memory space for the data sets to be used in the system, as the choice and format of data affects performance of the algorithm.

Scope of possible parallel computational processing is discussed as follows: Harvey et al. (Harvey et al., 2008) and Anderson et al. (Anderson and Coupland, 2008) in their respective work have given a vast scope of parallelism for Type-1 FLS. In the same fashion Ngo et al. (Ngo et al., 2012) presented a novel scope of parallelism for Interval Type-2 FLS. However, in all these implementations the emphasis was laid to parallelize the number of fuzzy rules and discrete levels for a typical FLS with two inputs and single output. So, a single FLS was computed with parallel rule inference on GPU using CUDA. However, a typical FLS can be processed multiple times with fixed fuzzy rules and discrete levels but varying inputs. Here lies our scope of parallelism, we construe our code to compute multiple FLS in parallel on GPU as a FLS serially on CPU will consume more time.

6.3 Scope of Parallelism in Type-2 FLS

Generalized T2 FLSs are computationally more intensive as compared to T1 FLS as former includes FSs those are 3-dimensional in nature. Things do simplify when secondary membership functions are considered as interval sets, i.e., the secondary membership values are either 0 or 1 and set are referred as Interval Type-2 FSs or simply IT2 FSs. IT2 FSs have received the most investigational interests as they involve mathematics that is simpler than that of generalized T2 FSs. Therefore, literature available about IT2 FSs is more as compared to that of generalized T2 FSs. Now a days, both kinds of fuzzy sets are being actively investigated by an ever-growing number of researchers around the world.

IT2 FSs have widely been accepted as they provide more freedom degree in modelling higher orders of uncertainty than T1 FSs. This property has been the driving force behind more of the advancements in theories and applications of IT2 FSs and FLSs. IT2 FSs are represented by upper and lower bounds of uncertainty called Upper Membership Function (UMF) and Lower Membership Function (LMF) as shown in Fig. **??** The region between upper and lower bounds of Uncertainty is termed as Footprint of uncertainty (FOU).

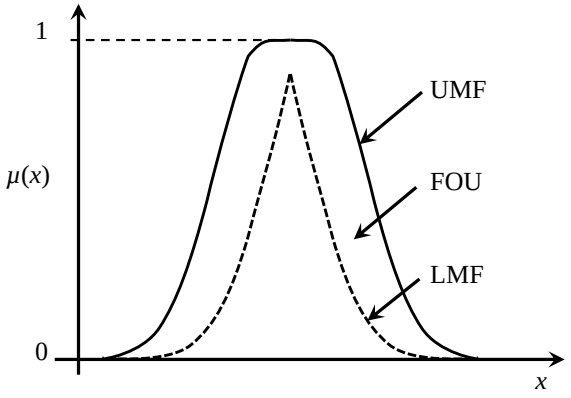

FIGURE 6.1: Interval Type-2 Fuzzy Set

According to theory of FLS given by Zadeh a fuzzy set is defined for a particular domain, and it is characterized by a membership function that maps elements from the domain to a real valued numbers (Zadeh, 1965b). Jerry Mendel and his students have reported numerous methods to design and implement FLSs for various applications (Karnik and Mendel, 1999; Mendel, 1995, 2000; Mouzouris and Mendel, 1997) based on the IT2 FSs as proposed by Zadeh in 1975 (Zadeh, 1975).

In this work, every input has been fuzzified using four Gaussian shaped IT2 FSs. Each IT2 FS is designed using two gaussian shaped T1 FSs, ($\mu_L(x)$ and $\mu_R(x)$) having same standard

variance and different means (m_L and m_R). Mathematically, given as (6.3)

$$\mu_L(x) = \exp\left(\frac{-(x-m_L)^2}{2\sigma^2}\right) \text{ and } \mu_R(x) = \exp\left(\frac{-(x-m_R)^2}{2\sigma^2}\right) \tag{6.3}$$

Further, UMF and LMF are determined using union and intersection of these left and right T1 FSs as given by (6.4). Here, upper and lower bounds of LMF and UMF represent the uncertainty in an IT2 antecedent FS, A, as shown in Fig. 6.1.

$$\underline{\mu}_A(x) = \mu_L(x) \cap \mu_R(x) \text{ and } \overline{\mu}_A(x) = \mu_L(x) \cup \mu_R(x) \tag{6.4}$$

In this work, an IT2 FLS with four inputs and one output is subjected to parallel computation for forecasting of Mackey-Glass time series (Karnik and Mendel, 1999) (Khosla et al., 2011). Consequents of the FLS are considered singleton interval sets, represented as $[C_{Li}\ C_{Ri}]$ for $i = 1, 2, ...$ For implication and aggregation operations, standard *min* and *max* fuzzy operator are experimented. The defuzzification is performed using the height defuzzification method. As all involved FSs are symmetrical, therefore, response of height defuzzification method will be equivalent to that of center of gravity method of defuzzification. Mathematically, after firing M number of fuzzy rules the aggregated fuzzy set, B, is subjected to defuzzification and the crisp output is determined using (6.5)-(6.7)

$$y_L = \frac{\sum_{i=1}^{M} C_{Li}\underline{\mu}_B(y)}{\sum_{i=1}^{M} \underline{\mu}_B(y)} \tag{6.5}$$

$$y_U = \frac{\sum_{i=1}^{M} C_{Ri}\overline{\mu}_B(y)}{\sum_{i=1}^{M} \overline{\mu}_B(y)} \tag{6.6}$$

$$y = \frac{1}{2}(y_L + y_U) \tag{6.7}$$

Scope of possible parallel computational processing is discussed as follows: Harvey et al. (Harvey et al., 2008) and Anderson et al. (Anderson and Coupland, 2008) in their respective work have investigated T1 FLSs, with two inputs and single output, for parallelism in firing multiple fuzzy rules and handling increased sample rate of universe of discourses for both antecedents and consequents. On the similar lines, Ngo et al. (Ngo et al., 2012) have presented a novel scope of parallelism for IT2 FLSs. However, in this work, a typical FLS having four antecedents and single consequents has been investigated to run multiple fuzzy rules and increased input data sets in parallel.

6.4 Parallel Type-1 Fuzzy Logic System

Two $M \times N$ dimensional 'Mean' and 'Sigma' matrices are used in this implementation where M denotes number of fuzzy rules and N is number of antecedents. These matrices hold mean and sigma values of Gaussian membership function in accordance with fuzzy rules used in the FLS. An $M \times 1$ dimensional 'Consequent' matrix contains only mean values of the consequent fuzzy sets as systems uses height defuzzification method given by equation (6.2). Multiple inputs are provided to the systems collectively in the form of an $L \times N$ dimensional input matrix, X, where L is the number of inputs.

The CPU executes rule firing sequentially with a single input at a time and causes more computational time. Whereas, multiple threads are run at the same time to fire multiple rules simultaneously using system matrices, i.e., 'Mean', 'Sigma' and 'Consequent' matrices, that reduce the execution time for the FLS. A kernel function is initialized from CPU to pass inputs in parallel to various GPU cores. Copying system matrices everytime along with input vectors and increases the GPU processing time. Therefore, system matrices are copied only once and subsequently requires only input vectors those are passed to GPU in parallel to enhances the GPU performance.

6.5 Parallel Interval Type-2 Fuzzy Logic System

Three $M \times N$ dimensional 'Mean-1', 'Mean-2' and 'Sigma' matrices are used in this implementation of IT2 FLS on CPU and GPU where M denotes number of fuzzy rules and N is number of antecedents. These matrices hold mean and sigma values of Gaussian membership function in accordance with M fuzzy rules used in the Interval T2 FLS. Two $M \times 1$ dimensional 'Consequents' matrices contains only mean values of the consequent fuzzy sets as system uses height defuzzification method given by equation (6.5)-(6.7). Multiple inputs are provided to the systems, collectively, in the form of an $L \times N$ dimensional input matrix, X, where L is the number of inputs or size of the input data set.

The CPU executes rule firing sequentially with a single input at a time and causes more computational time. Whereas, multiple threads are run at the same time to fire multiple rules simultaneously using system matrices, i.e., 'Mean-1', 'Mean-2', 'Sigma' and two 'Consequents' matrices, that reduce the execution time for the IT2 FLS. A kernel function is initialized from CPU to pass inputs in parallel to various GPU cores. Copying system matrices every time along with input vectors and increases the GPU processing time. Therefore,

system matrices are copied only once and subsequently requires only input vectors those are passed to GPU in parallel to enhance the GPU performance.

6.6 Conclusion

CUDA implementation is easier for T1-FLS as compared to IT2-FLS. IT2-FLS carries more computation than T1-FLS and moreover it is dense in calculations but according to GPU's power of paallelism more is the computation more better the GPU performs with the CUDA platform. CUDA implementation in this work is laid on the matrix manipulation of the data sets in to the unique mathematics so that efficient parallelism can be obtained to improve the results of speedup.

CHAPTER 7

RESULTS AND DISSCUSSIONS

In this chapter, the fuzzy system modules are discussed which have been implemented so far in C language which is a serial implementation of the fuzzy logic system. The system comprises of various structures and functions which are used in the code for fuzzification and implementation of the same.

7.1 Introduction

The speed up performance with GPU implementation of T1 Sugeno FLS and IT2 FLS was compared with that of CPU implementations respectively. Intel Core 2 Duo system under experimentation has 2GB of system RAM, and Windows 7 platform. The GPU used here works on nVIDIA Geforce GTX 650 with 1024 MB of texture memory, 192 stream processors, and PCI Express X16.

7.2 Performance Analysis of Type-1 FLS

The number of antecedents were fixed to 4 and consequent to 1, the number of rules were varied between 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 and inputs were varied as 128, 256, 512, and 1024. A set of input data is obtained from Mackey-Glass time series for experimentation. Ratio of CPU to GPU run times with respect to number of fuzzy rules has been tabulated

in Table 7.1 and presented graphically in Fig. 7.1. Here, it can be observed clearly that advantages of GPU computing increases with the increased input data vector sizes.

TABLE 7.1: Fuzzy Rules vs CPU to GPU Speedup ratio (T1 FLS)

Number of Fuzzy Rules	CPU to GPU Runtime Speedup Ratio			
	128 Inputs	256 Inputs	512 Inputs	1024 Inputs
10	1.937	4.875	5.034	5.488
20	2.437	3.032	4.548	6.132
30	1.340	2.319	4.319	6.544
40	2.000	2.476	3.968	7.063
50	1.516	3.532	4.410	7.063
60	1.730	3.000	4.365	7.185
70	1.602	3.205	4.509	7.134
80	1.500	2.742	4.500	7.832
90	1.709	2.577	4.446	5.488
100	1.624	2.504	4.652	6.132

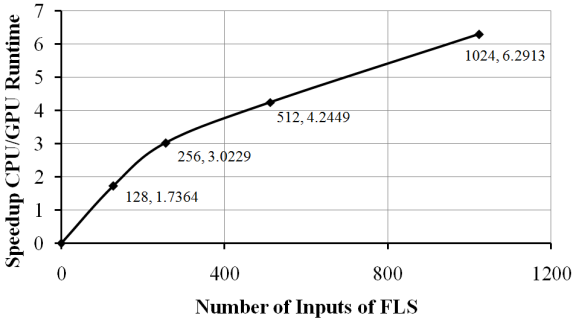

FIGURE 7.1: CPU to GPU speedup ratio Vs Inputs data length (T1 FLS)

In another simulation results, it is observed that CPU to GPU speedup time improves with increase in fuzzy rules. Computational time on CPU varies significantly due to already always running applications at the back end. Therefore, to present fair comparison serial and parallel timing analysis experiments with same FLS have been repeated 30 times. Average of CPU to GPU speedups for 30 monte-carlo simulations during serial and parallel computations of rule firing, implication, aggregation and defuzzification have been presented in Fig. 7.2-7.5.

The overall performance of CPU to GPU speedup timings with respect to collective number of inputs, presented to the FLS, is shown in Fig. 7.6 that depicts that larger the number of fuzzy rules better is the speedup performance.

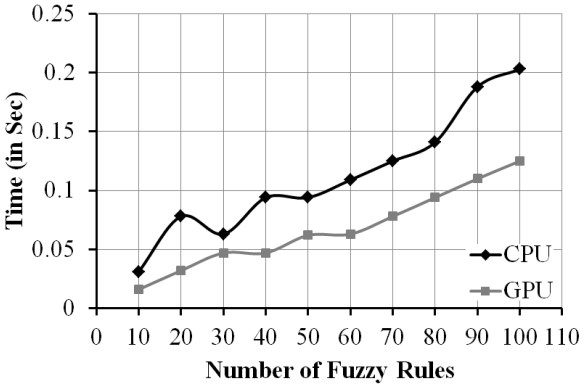

FIGURE 7.2: CPU and GPU Runtime comparison for 128 inputs (T1 FLS)

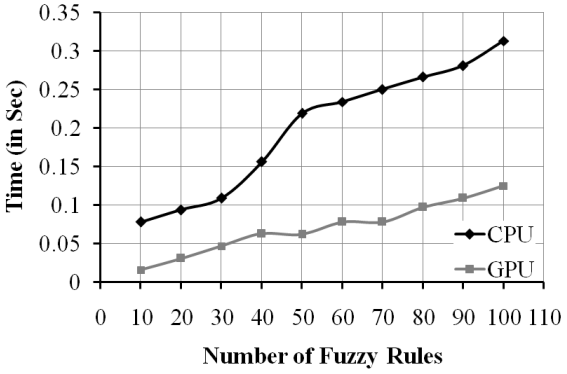

FIGURE 7.3: CPU and GPU Runtime comparison for 256 inputs (T1 FLS)

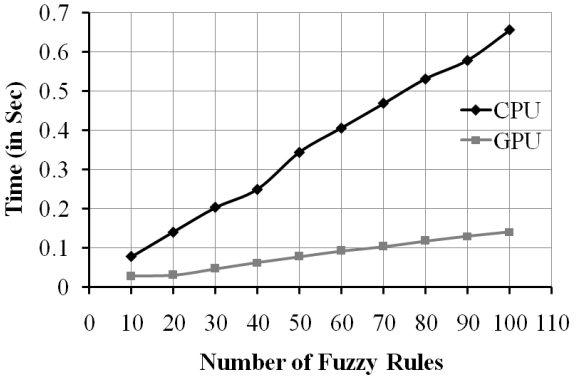

FIGURE 7.4: CPU and GPU Runtime comparison for 512 inputs (T1 FLS)

7.3 Performance Analysis of Interval Type-2 FLS

The number of antecedents are kept to four and single consequent, the number of rules are varied as 10, 20, 30, ..., 100 and inputs are varied as 128, 256, 512, and 1024. A set of input data is obtained from Mackey-Glass Time Series for experimentation. Ratio of CPU to GPU runtimes with respect to number of fuzzy rules has been tabulated in Table 7.2 and presented graphically in Fig. 7.7. Here, it can be observed that advantages of GPU

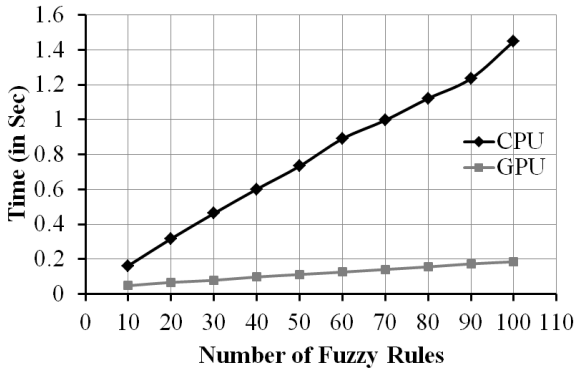

FIGURE 7.5: CPU and GPU Runtime comparison for 1024 inputs (T1 FLS)

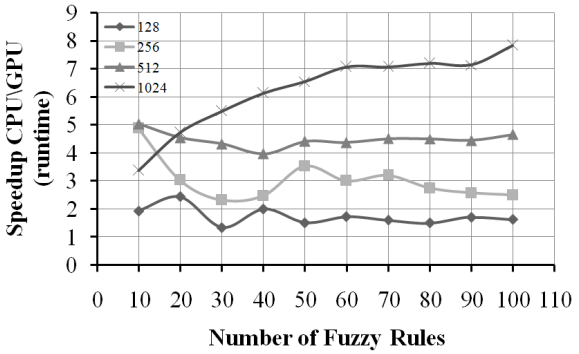

FIGURE 7.6: CPU to GPU Speedup ratio for various input data sets (T1 FLS)

computing increases with the increased input data vector sizes. It can be also observed here

TABLE 7.2: Fuzzy Rules and number of Inputs vs CPU to GPU Speedup ratio (IT2)

Number of Fuzzy Rules	**CPU to GPU Runtime Speedup Ratio**			
	128 Inputs	**256 Inputs**	**512 Inputs**	**1024 Inputs**
10	1.777	3.200	4.807	8.258
20	1.969	3.378	6.097	10.541
30	1.958	3.528	6.147	10.619
40	2.062	3.764	6.204	11.108
50	2.000	3.771	6.237	11.933
60	2.010	3.858	6.978	11.183
70	2.036	3.863	7.302	11.159
80	2.064	3.873	7.605	11.244
90	2.014	3.903	7.641	11.750
100	2.006	4.04	7.725	12.560
Average Speedup	**1.9896**	**3.7178**	**6.6743**	**11.0355**

that CPU to GPU speedup time improves with increase in fuzzy rules (vertically downwards in Table. 7.2). Computational time on CPU varies significantly due to already always running applications at the back end. Therefore, to present fair comparison serial and parallel timing analysis experiments with same FLS have been repeated 30 times. Average of CPU to GPU

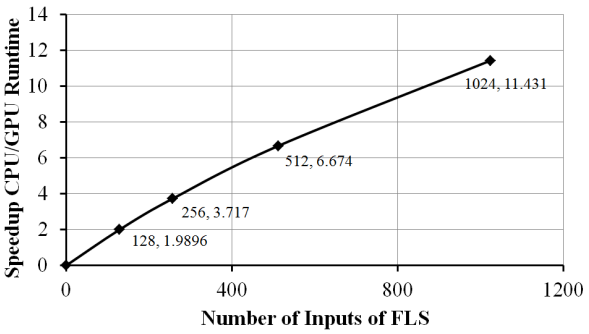

FIGURE 7.7: Average CPU to GPU Speedup ratio Vs Input data sizes for IT2 FLS

speedups for 30 monte-carlo simulations during serial and parallel computations of rule firing, implication, aggregation and defuzzification have been presented in Fig. 7.8-7.11.

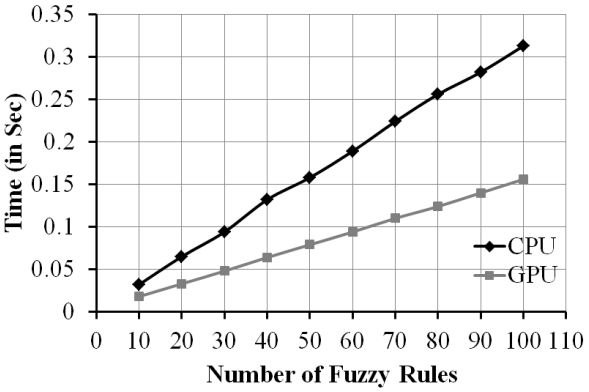

FIGURE 7.8: CPU and GPU Runtime comparison for 128 inputs (IT2 FLS)

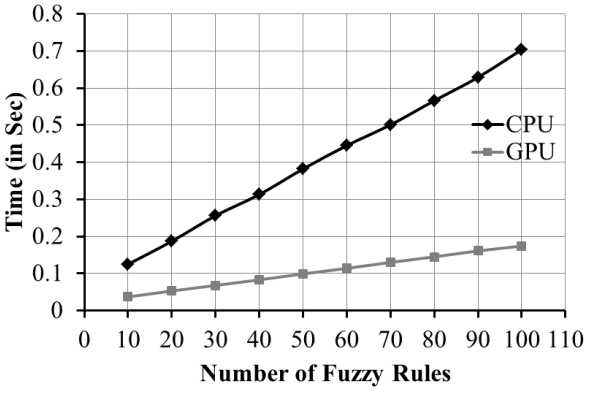

FIGURE 7.9: CPU and GPU Runtime comparison for 256 inputs (IT2 FLS)

The overall performance of CPU to GPU speedup timings with respect to collective number of inputs, presented to the FLS, is shown in Fig. 7.12 that depicts, larger the number of fuzzy rules better is the speedup performance.

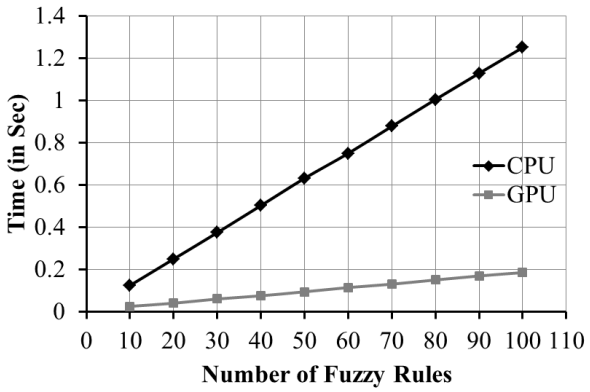

FIGURE 7.10: CPU and GPU Runtime comparison for 512 inputs (IT2 FLS)

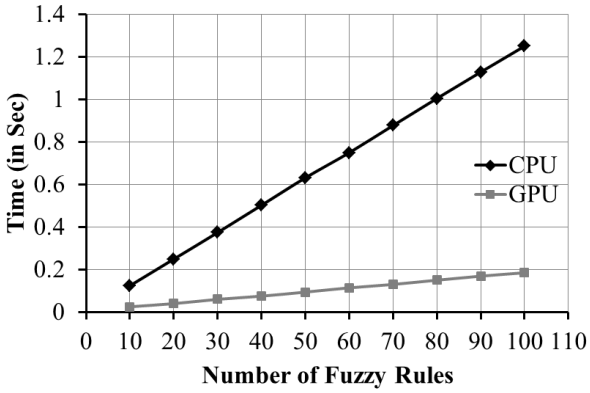

FIGURE 7.11: CPU and GPU Runtime comparison for 1024 inputs (IT2 FLS)

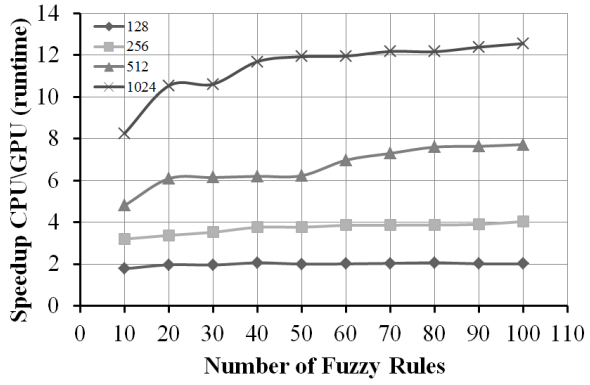

FIGURE 7.12: CPU to GPU Speedup ratio for various input data sets (IT2 FLS)

7.4 Overall Performance

This chapter has two conclusions to make which are given below:

7.4.1 Performance of T1 FLS

This work has demonstrated the implementation and comparative runtime performances of a typical Sugeno Type-1 FLS on a GPU and CPU without the use of a graphics API which is flexible, scalable, and can be used by any researcher with knowledge of C. It has been demonstrated that the CPU works equally fast as GPU when the system is small. As the number of rules or the number of inputs increase the GPU outperforms the CPU runtime. Here, in the work nearly 7.83 times speedup could be achieved as 1024 inputs supplied in parallel to the FLS ported on GPU. The GPU has an initial setup overhead of kernel loading and memory transfer, however, subsequent parallel computations leads to a small increase in processing time despite a substantial increase in computational load. On the other hand, CPU has no initial cost, but computation time grows linearly with computational load much beyond GPGPU runtime.

7.4.2 Performance of IT2 FLS

This work has demonstrated the implementation and comparative runtime performances of a typical IT2 FLS on a GPU and CPU without the use of graphics API which is flexible, scalable, and can be used by any researcher with abstract knowledge of C. It has been demonstrated that the CPU works equally fast as GPU when the system is small. As the number of rules or the number of inputs increase the GPU outperforms the CPU runtime. Here, in the work maximum of 12.56 times speedup could be achieved as 1024 inputs supplied in parallel to the IT2 FLS ported on GPU. The GPU has an initial setup overhead of kernel loading and memory transfer, however, subsequent parallel computations leads to a small increase in processing time despite a substantial increase in computational load. On the other hand, CPU has no initial cost, but computation time grows linearly with computational load much beyond GPGPU runtime.

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

This chapter, presents overall conclusion of the research work carried out in the thesis. It also gives off-shoots those are on our future agenda.

8.1 Introduction

This thesis has been organized in order to achieve speedup performance of running and computing FLSs in parallel. In the results and discussions it is clear that CUDA has been successfully able to achieve speedup on GPU whereas the same implementation in serial on CPU is slow the detailed conclusion of both T1-FLS and IT2-FLS is given below:

8.2 T1 FLS

This work has demonstrated the implementation and comparative runtime performances of a typical T1 FLS on a GPU and CPU without the use of a graphics API which is flexible, scalable, and can be used by any researcher with abstract knowledge of C. It has been demonstrated that the CPU works equally fast as GPU when the system is small. As the number of rules or the number of inputs increase the GPU outperforms the CPU runtime.

Here, in the work maximum of 7.83 times speedup could be achieved as 1024 inputs supplied in parallel to the T1 FLS ported on GPU. The GPU has an initial setup overhead of kernel loading and memory transfer, however, subsequent parallel computations leads to a small increase in processing time despite a substantial increase in computational load. On the other hand, CPU has no initial cost, but computation time grows linearly with computational load much beyond GPGPU runtime.

8.3 IT2 FLS

This work has demonstrated the implementation and comparative runtime performances of a typical IT2 FLS on a GPU and CPU without the use of graphics API which is flexible, scalable, and can be used by any researcher with abstract knowledge of C. It has been demonstrated that the CPU works equally fast as GPU when the system is small. As the number of rules or the number of inputs increase the GPU outperforms the CPU runtime. Here, in the work maximum of 12.56 times speedup could be achieved as 1024 inputs supplied in parallel to the IT2 FLS ported on GPU. The GPU has an initial setup overhead of kernel loading and memory transfer, however, subsequent parallel computations leads to a small increase in processing time despite a substantial increase in computational load. On the other hand, CPU has no initial cost, but computation time grows linearly with computational load much beyond GPGPU runtime.

8.4 Future work

The parallelization of more FLS applications is next on our agenda. That will follow implementation of Generalized T2 FLSs for various applications that require much more computational time otherwise. GPGPU is also possibly investigated on Evolutionary Algorithms those are computational intensive and parallel in nature.

8.5 Conclusion

This thesis has provided a vast knowledge of the research area of parallel computing and its combination with AI algorithms is really beneficial so far. FLSs are really the right contender parallel computation on GPU. The considerable speedup in the work has proved this statement true and the unique idea of running multiple FLS in parallel will be beneficial

for future works as in implementation of MGTS it will be really beneficial. In practice of such FLS for realtime applications (either T1 or IT2 FLS) one needs multiple FLS running in parallel for speedup performance of the application. So, the work in this thesis has given a new and right method for computing such applications in efficient and fast way using GPU with CUDA programming platform. And these systems prove to be more efficient when there is a least memory transfer from CPU to GPU and vice versa during code working. In the future GPGPU and CUDA applications has a bright future ahead.

REFERENCES

Anderson, D. and Coupland, S. (2008). Parallelisation of fuzzy inference on a graphics processor unit using the compute unified device architecture. In *Proceedings of the UK Workshop on Computational Intelligence (UKCI'08)*, pages 1–6.

Anderson, D. T., Luke, R. H., and Keller, J. M. (2008). Speedup of Fuzzy Clustering Through Stream Processing on Graphics Processing Units. *Fuzzy Systems, IEEE Transactions on*, 16(4):1101–1106.

Arora, R., Tulshyan, R., and Deb, K. (2010). Parallelization of Binary and Real-Coded Genetic Algorithms on GPU Using CUDA. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

Boyer, M., Tarjan, D., Acton, S. T., and Skadron, K. (2009). Accelerating Leukocyte Tracking Using CUDA: A Case Study in Leveraging Manycore Coprocessors. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12. IEEE.

Christophe, E., Michel, J., and Inglada, J. (2011). Remote Sensing Processing: From Multicore to GPU. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 4(3):643–652.

da Silva, A. F. (2010). cudaBayesreg: Bayesian Computation in CUDA. *The R Journal*, 2(2):48–55.

Delévacq, A., Delisle, P., Gravel, M., and Krajecki, M. (2013). Parallel ant colony optimization on graphics processing units. *Journal of Parallel and Distributed Computing*, 73(1):52–61.

Harvey, N., Luke, R., Keller, J. M., and Anderson, D. (2008). Speedup of fuzzy logic through stream processing on graphics processing units. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3809–3815. IEEE.

Hong-Tao, B., Li-li, H., Dan-tong, O., Zhan-shan, L., and He, L. (2009). K-means on ommodity GPUss with Cuda. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, volume 3, pages 651–655. IEEE.

Karnik, N. N. and Mendel, J. M. (1999). Applications of type-2 fuzzy logic systems to forecasting of time-series. *Information Sciences*, 120(1):89–111.

Khosla, M., Sarin, R. K., Uddin, M., Khosla, A., and Singh, S. (2011). *Realizing Interval Type-2 Fuzzy Systems with Type-1 Fuzzy Systems*. IGI (USA).

Liang, Q. and Mendel, J. M. (2000). Interval type-2 fuzzy logic systems: theory and design. *Fuzzy Systems, IEEE Transactions on*, 8(5):535–550.

Liu, W., Schmidt, B., Voss, G., and Müller-Wittig, W. (2008). Accelerating Molecular Dynamics Simulations Using Graphics Processing Units with CUDA. *Computer Physics Communications*, 179(9):634–641.

Mendel, J. M. (1995). Fuzzy Logic Systems for Engineering: A Tutorial. *Proceedings of the IEEE*, 83(3):345–377.

Mendel, J. M. (2000). Uncertainty, fuzzy logic, and signal processing. *Signal Processing*, 80(6):913–933.

Mendel, J. M. and Mouzouris, G. C. (1997). Designing Fuzzy Logic Systems. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 44(11):885–895.

Mielikainen, J., Huang, B., Huang, H., and Goldberg, M. D. (2012). Improved GPU/CUDA Based Parallel Weather and Research Forecast (WRF) Single Moment 5-Class (WSM5) Cloud Microphysics. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 5(4):1256–1265.

Mouzouris, G. C. and Mendel, J. M. (1997). Nonsingleton fuzzy logic systems: theory and application. *Fuzzy Systems, IEEE Transactions on*, 5(1):56–71.

Mundici, D. and Olivetti, N. (1998). Resolution and model building in the infinite-valued calculus of lukasiewicz. *Theoretical Computer Science*, 200(1):335–366.

Mussi, L., Cagnoni, S., and Daolio, F. (2009). GPU-Based Road Sign Detection Using Particle Swarm Optimization. In *Intelligent Systems Design and Applications, 2009. ISDA'09. Ninth International Conference on*, pages 152–157. IEEE.

Ngo, L. T., Nguyen, D. D., Luong, C. M., et al. (2012). Speedup of interval type 2 fuzzy logic systems based on GPU for robot navigation. *Advances in Fuzzy Systems*, 2012:4.

Nuli, U. A. and Kulkarni, P. (2012). Sph Based Fluid Animation Using Cuda Enabled GPU. *International Journal of Computer Graphics and Animation*.

Schulz, C. (2013). Efficient Local Search on the GPUInvestigations on the Vehicle Routing Problem. *Journal of Parallel and Distributed Computing*, 73(1):14–31.

Shi, Y., Eberhart, R., and Chen, Y. (1999). Implementation of Evolutionary Fuzzy Systems. *Fuzzy Systems, IEEE Transactions on*, 7(2):109–119.

Thompson, E. A. and Anderson, T. R. (2012). Use of CUDA for the Continuous Space Language Model. In *HPEC*, pages 1–5.

Zadeh, L. A. (1965a). Fuzzy sets. *Information and control*, 8(3):338–353.

Zadeh, L. A. (1965b). Fuzzy Sets. *Information and control*, 8(3):338–353.

Zadeh, L. A. (1975). Fuzzy Logic and Approximate Reasoning. *Synthese*, 30(3-4):407–428.

INDEX

Base Papers, 7

Bayesian Computation, 5

Continuous Space Language Model, 6

CUDA, 22

- Architecture, 23
- Programming Model, 24

Fuzzy Logic, 10

GPGPU, 15, 22

GPU, 15

- Computing, 19

Interval Type-2 Fuzzy Logic System, 13

K-Mean Computation, 5

Leukocyte Tracking, 5

Mackey-Glass time series, 30

Methodology, 2

Molecular Dynamics Simulation, 6

Motivation, 2

nVIDIA, 1

Performance

- Overall, 40
- Type-1 FLS, 35
- Type-2 FLS, 37

Remote Sensing, 5

Research Gaps, 9

SIMD, 20, 25

SPH, 4

Sugeno Type-1 FLS, 41

Thesis

- Conclusion, 43
- Introduction, 1
- Objectives, 2
- Outline, 3

Type-1 Fuzzy Logic System, 11

Type-2 Fuzzy Logic System, 12

Vehicle Routing, 6

Weather Forecasting, 5