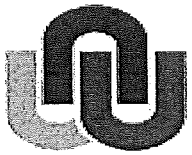

FACULTY OF AGRICULTURE SCIENCE AND TECHNOLOGY

Optimized dynamic programming search for automatic speech recognition on a Graphics Processing Unit (GPU) platform using Compute Unified Device Architecture (CUDA)

By: Babedi Betty Letswamotse

North-West University

(Mafikeng Campus)

This dissertation is submitted to the Department of Computer Science in fulfilment for the requirements for MSc in Computer Science.

Supervisor: Dr Naison Gasela

North-West University (Mafikeng Campus)

Co-supervisor: Dr Z.P Ncube

Sol Plaatjie University

May 2014

APPROVAL

This project has been submitted for examination with my approval as the candidate's University supervisor.

Signature: ..... Date: .....

Dr. Naison Gasela

DECLARATION

I **Betty Babedi Letswamotse**, hereby declare that the research project entitled Optimized dynamic programming search for automatic speech recognition on a GPU platform using CUDA is entirely my own work, except where acknowledged, and it has never been submitted to any other university or institution of higher learning for the award of a degree.

Signature: BLetswamotse.....

Betty Babedi Letswamotse

Date: 29-09-2014.....

ACKNOWLEDGEMENT

Firstly, I would like to express my gratitude to my supervisor Dr Naison Gasela and My co-supervisor Dr Z.P Ncube for their support and constructive criticism during the course of this research. My gratitude to the Faculty of Agriculture, Science and Technology; North-West University (Mafikeng Campus) for the support they have given me in ensuring that I complete my degree in record time.

I would also like to express my warm appreciation to my lab partner Mr Kgotlaetsile Modieginyane for all the laughter, the encouragement and for all the sleepless nights we were working together prior to the deadlines.

Furthermore, I would like to thank my family for their support and motivation.

Last but not the least; I would like to thank God almighty for the strength and courage.

Abstract

In a typical recognition process, there are substantial parallelization challenges in concurrently assessing thousands of alternative interpretations of a speech utterance to find the most probable one. During this process, uttered words are converted into fragments. Decoding these fragments to produce relevant output is a computationally expensive task. To optimize Viterbi search requires a certain level of parallelism since search is a parallel process. We find that a better way to optimize speech recognition search is by the use of parallel architectures such as graphic processing units (GPUs). GPUs provide large computational power at a very low expense which positions them as viable global accelerators. We implemented the speech recognition Viterbi search algorithm on CPU and GeForce 8800 GTX GPU based systems in three implementations. The first implementation was implemented on a CPU based system, and then the original Viterbi search algorithm with the application of loop unrolling was implemented on the CPU based system and also on the GPU based systems. The GPU optimised implementation achieved a 30× speedup over the original CPU implementation and 8× speedup over the CPU implementation with the application of loop unrolling whereas the CPU implementation with the application of loop unrolling achieved a 4× speedup over the original CPU implementation of the Viterbi search algorithm. Achievements from our GPU optimized implementation have positively impacted on the overall speech recognition accuracy, thereby contributing across the field of automatic speech recognition.

Table of Contents

List of Figures and Tables

List of Figures

List of Tables

List of Acronyms

ANN	Artificial Neural Networks
ARM	Advanced Reduced Instruction Set Computing Machines
ASD	Attention Shift Decoding
ASR	Automatic Speech Recognition
BBN	Bolt Beranek Newman
CMU	Carnegie Mellon University
CRF	Condensation Resistance Factor
CSJ	Corpus Juris Secundum
CSR	Centre for Science Review
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DARPA	Defence Advanced Research Projects Agency
DCT	Discrete Cosine Transform
DHMM	Discrete Hidden Markov Model
DSR	Data Signal Rate
DTW	Dynamic Time Warping
EARS	Effective Affordable Reuse Speech-to-text
EM	Expectation Maximization
EPPS	European Parliamentary Plenary Session
FSN	Finite State Network
GB	Gigabyte

GB/s	Gigabyte per second
GHz	Gigahertz
GMM	Gaussian Mixture Models
GPGPU	General Purpose Computing on GPUs
GPU	Graphics Processing Unit
HMM	Hidden Markov Model
HDR	High Dynamic Range
HPC	High Performance Computing
HWIM	Hear What I Mean
HTK	Hidden Markov Model Toolkit
IBM	International Business Machines
LPC	Linear Predictive Coding
LVCSR	Large Vocabulary Continuous Speech Recognition
MATLAB	Matrix Laboratory
MB	Megabyte
MCE	Machine Check Exception
MFCC	Mel Frequency Cepstral Coefficients
MIT	Massachusetts Institute of Technology
MLP	Multilayer Perceptron
MPI	Message Passing Interface
NAB	National Association of Broadcasters
NEC	Nippon Electronic Company
PLP	Perceptual Linear Prediction

POSIX	Portable Operating System Interface
RCA	Radio Corporation of America
RTF	Real Time Factor
SME	Soft Margin Estimation
SMFE	Soft Margin Feature Extraction
SSE	Streaming SIMD Extensions
SWER	Single Word Error Rate
SWIFT	Speedy Weighted Finite State Transducers
TC-STAR	Technology and Corpora for Speech to Speech Translation
TRBF	Temporal Radial Basis Function
UCL	University College London
UTA	University of Texas
WAR	Word Accuracy Rate
WER	Word Error Rate
WFST	Weighted Finite State Transducers

Chapter 1: Introduction

This chapter discusses the general introduction to the research. The problem statement, research questions, aims and objectives are presented. A substantiation of the study and summary of the dissertation, are given.

1.1. Introduction to Research

Speech is the most effective form of communication for human to human interactions, so people expect the same when it comes to human-machine (computer) interactions. They expect speech recognition systems in which the computer speaks and recognizes any human language. For these expectations to be met, speech recognition has to be put into practice. Speech recognition is the process of recognizing spoken input and converting it into written text through a speech recognition system.

In recent years there have been tremendous advancements in the field of speech recognition. Speech technology is rapidly growing and it is used commercially to provide services such as telephone directory assistance, telephone shopping and banking services among other applications. In the Education sector it is used for foreign language translation whereby speech (uttered words) is automatically translated into non-native language(s).

Speech recognition systems are also used in a wide range of applications such as in air traffic control, embedded telecommunication systems, robotics, computer and video games, and also to help people with disabilities e.g. blind people and the physically disabled people. The performance of Speech recognition systems is usually assessed by means of accuracy (Word Error Rate (WER)) and speed (Real Time Factor (RTF)). Most of the modern speech recognition systems are usually based on statistical models such as Hidden Markov Models (HMMs). According to [**1**], HMMs are popular due to their simplicity and are computationally less intensive and parameters that can be estimated automatically from a large amount of data.

Speech recognition is divided into two stages: feature extraction and classification. The classification stage is a collection of segmented words and sub-words into different classes based on some properties [**2**]. Classification comprises acoustic models which are files that are generated by taking audio recordings of speech and their transcriptions and then compiling them into statistical representations of the sounds for words. Each of these

statistical representations is assigned a label called a phoneme [3]and [4]. There is a pronunciation dictionary which is a machine-readable dictionary that contains a collection of words and their transcriptions and a language model which is a probability distribution P(s) over words S that attempts to reveal how frequently a string S occurs as a sentence. Language models are often used for dictation applications. In any speech recognition system the two vital metrics to contemplate include the elapsed time between the acquisition of the speech signal and the recognized word.

In a typical recognition process, there are substantial parallelization challenges in concurrently assessing thousands of alternative interpretations of a speech utterance (a natural unit of speech bounded by breaths or pauses [5]) to find the most probable interpretation. Many time critical applications are unable to use Automatic Speech Recognition (ASR) due to the heavy latency in processing the speech with a large vocabulary size.

In this research the author focuses on the decoding process of speech recognition. The decoding process which is often referred to as search in a speech recognizer's operation is to find a sequence of words whose corresponding acoustic and language models best match the input feature vector sequence [1]. Search is a computationally expensive task since it handles irregular graph structures with data parallel operations. There are algorithms that were developed specifically for this task but many search algorithms were developed long prior to the existence of parallelism.

Dynamic Programming is the foundation of most broadly used speech recognition systems. Search strategies based on dynamic programming are currently being used successfully for a large number of speech recognition tasks. These tasks range from digit string recognition through medium-size vocabulary recognition using heavily constrained grammars, to large vocabulary continuous speech recognition (LVCSR) with virtually unconstrained speech input [6].

Optimizing dynamic programming search for automatic speech recognition has been studied previously. To optimize dynamic programming search requires a certain level of parallelism since search is a parallel process. A better way to optimize speech recognition search is using parallel architectures such as graphic processing units (GPUs). GPU is a specialized circuit designed to accelerate the image output in a frame buffer intended for output to a display. GPUs are very efficient at manipulating computer graphics and are generally more

effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel [7]. GPUs provide large computational power at a very low expense which positions them as global accelerators. These savings encourage using GPUs as hardware accelerators to support computationally intensive applications.

The introduction of parallel programming languages such as CUDA, OpenCL, MPI, OpenMP and others made general purpose computing a lot easier on the GPU side. CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA to enable efficient computing performance by harnessing the power of the graphics processing unit so that it is a scalable parallel programming model and a software environment for parallel computing [7]. The author proposes using Viterbi beam search for this research. Beam search is a heuristic method for combinatorial optimisation problems, which has been studied [8] widely in artificial intelligence and operations research. It is related to breadth-first search as it progresses level by level through a highly structured search tree containing all probable solutions to a problem, but it does not explore all the encountered nodes.

1.2. Background

This section presents the general speech recognition theory and history.

1.2.1. Theory of Speech Recognition

A speech recognition system could either be an embedded application or a combination of an embedded application and a computer. The main component of the speech recognition system is the speech recognition engine, which is the software that translates a spoken signal into text.

Speech recognition is divided into two phases. The first phase is feature extraction and the second phase is classification and decoding. Feature extraction techniques extract acoustic features from the speech waveform. The most widely used acoustic feature sets for Automatic Speech Recognition (ASR) are Mel-frequency cepstral coefficients (MFCCs) and Perceptual Linear Prediction (PLP) coefficients. When using MFCC the input waveform signal goes through a Mel-scaled filter bank first. Then it is followed by low pass filtering and down sampling. Lastly discrete cosine transform (DCT) is performed on the log-energy of the filter outputs [9]. The classification phase comprises acoustic models, pronunciation

dictionary and language models. Figure 2.1 depicts the components of the speech recognition system.

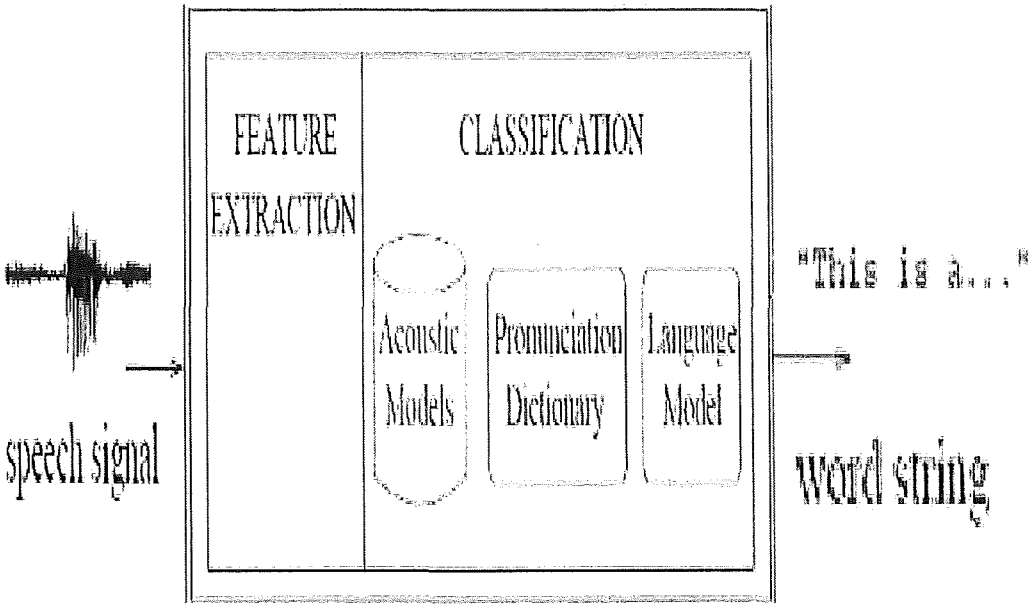

Figure 1.1: components of an ASR system adapted from [55]

In any speech recognition systems there are vital metrics to contemplate: the elapsed time between the acquisition of the speech signal and the recognized word and the recognition accuracy [**10**]. Speech recognition systems are classified according to factors such as the type of utterance, type of speaker model, type of channel and the type of vocabulary that it is able to recognise [**11**]. The methods and algorithms that have been used for different speech recognition systems thus far are classified into template based [**12**], model based [**13**], neural network based [**14**] and Discriminant Analysis Methods based on Bayesian discrimination.

1.2.2. History of Speech Recognition

Speech recognition dates back to the 1920s when the technology was still complex. The first invention was a radio toy which was able to recognize speech. The technology progressed throughout the years as researchers found interest in the field of speech recognition. In 1952 Bell Laboratories designed a speaker dependent isolated digit recognition system [**15**]. In an independent effort RCA laboratories tried to recognize ten distinct syllables of a single speaker, as embodied in ten monosyllabic words in 1956 [**16**].

According to [**17**], in 1959 UCL created a phoneme recognizer that is able to recognize four vowels and nine consonants. They increased the overall phoneme recognition accuracy for words consisting of two or more phonemes by incorporating statistical information concerning allowable phoneme sequences in English; their work marked the first use of statistical syntax at phoneme level in automatic speech recognition. In 1959, [**18**] the MIT Lincoln Laboratories devised a system which was able to recognize ten vowels.

In 1960 Radio Research Lab in Tokyo built a hardware vowel recognizer [**19**]. In 1962 Kyoto University [**20**] built a hardware phoneme recognizer using a hardware speech segmenter and a zero-crossing analysis of different regions of the input utterance. In 1963 NEC Laboratories built a hardware digit recognizer [**21**]. In 1964 RCA laboratories came up with the realistic solution to the problem of no uniformity of time scales in speech events [**22**].

In 1968 the Soviet Union proposed the use of dynamic programming methods (Dynamic Time Warping (DTW)) for time aligning a pair of speech utterances, including algorithms for connected word recognition [**23**]. At the same time Sakoe and Chiba [**24**] at the NEC laboratories started using dynamic programming techniques to solve the problem of no uniformity. In the late 1960's Reddy [**25**] at CMU conducted a research in the field of continuous speech recognition by dynamic tracking of phonemes.

The 1970s brought the dawn of isolated word recognition. Dynamic programming methods progressed successfully in speech recognition through the works of Sakoe and Chiba [**24**]. Bell Laboratories, [**26**] showed how the ideas of linear predictive coding (LPC) could be extended to speech recognition systems through the use of an appropriate distance measure based on LPC spectral parameters. There was a great motivation to LVCSR so IBM researchers started studying large vocabulary speech recognition for three distinct tasks, namely the New Raleigh language [**27**], the laser patent text language [**28**], and the office correspondence task, called Tangora.

AT&T Bell Labs researchers began a series of experiments aimed at making speaker-independent speech recognition systems [**29**]. An ambitious speech understanding project was funded by the Defence Advanced Research Projects Agency (DARPA), which led to many seminal systems and technologies [**30**]. One of the first demonstrations of speech understanding was achieved by CMU in 1973. Their Hearsay I system was able to use

semantic information to significantly reduce the number of alternatives considered by the recognizer.

CMU's Harpy system was shown to be able to recognize speech using a vocabulary of 1,011 words with reasonable accuracy [**31**]. One particular contribution from the Harpy system was the concept of graph search, where the speech recognition language is represented as a connected network derived from lexical representations of words, with syntactical production rules and word boundary rules. The Harpy system was the first to take advantage of a finite state network (FSN) to reduce computation and efficiently determine the closest matching string. Other systems developed under the DARPA's speech understanding program included CMU'S Hearsay II and BBN'S HWIM (Hear What I Mean) systems [**30**].

The 1980s research focus was mostly based on connected word recognition. There was a progression from the template based approach to the statistical modelling approach on which most speech recognition systems are based. One of the key technologies developed in the 1980s is the HMM approach and the application of neural networks. In the 1980s, a deeper understanding of the strengths and limitations as well as an understanding of the relationship of the neural network technology to classical pattern classification methods was achieved by [**14**], [**32**] & [**33**].

The DARPA community conducted research on large-vocabulary, continuous-speech recognition systems, aiming at achieving high word accuracy for a 1000-word database management task [**34**]. Major research contributions resulted from efforts at CMU with the SPHINX system [**29**], BBN with the BYBLOS system [**35**], SRI with the DECIPHER system [**36**], Lincoln Labs [**37**], MIT [**38**] and AT&T Bell Labs [**29**]. The DARPA program continued into the 1990s, with emphasis shifting to natural language front ends to the recognizer. Speech recognition technology was increasingly used within telephone networks to automate as well as enhance operator services [**34**].

In the 2000s as part of the DARPA program, the Effective Affordable Reusable Speech-to-Text (EARS) program was conducted to develop speech-to-text (automatic transcription) technology with the aim of achieving substantially richer and much more accurate output than before. Several projects were conducted in order to increase the performance of spontaneous (natural language) speech recognition because the raise in spontaneous speech recognition was vital for broadening speech recognition applications. In Japan, a 5-year

national project "Spontaneous Speech: Corpus and Processing Technology" was conducted. The world's largest spontaneous speech corpus, "Corpus of Spontaneous Japanese (CSJ)" consisting of approximately 7 million words, corresponding to 700 hours of speech, was built, and various new techniques were investigated [**34**].

1.3. Problem Statement and Motivation

This section discusses the problem statement and substantiation.

1.3.1 Problem Statement

Speech recognition search is a critical process which handles irregular graph structures with data parallel operations. However, handling these irregular graph structures is a communication intensive process since the speech recognition system has to internally deal with recurrent speech acquisition processes as a means of achieving efficient speech recognition. In a typical recognition process, there are substantial parallelization challenges in concurrently assessing thousands of alternative interpretations of a speech utterance to find the most probable interpretation; this task faces problems such as word errors or recognition errors. During this process, uttered words (input signal) are converted into fragments (feature vectors), thus decoding these fragments to produce relevant output (word sequence) is a computationally expensive task.

The use of GPUs in automatic speech recognition has brought great improvements in speech processing systems since these cores perform well in processes requiring data-parallel execution. Though, to achieve scalable data-parallel execution speech applications; the literature informs to look at other challenges such as:

- Elimination of redundant work when threads are accessing an unpredictable subset of the results based on the input.
- Conflict free reduction on graph traversal to implement Viterbi beam search algorithm.
- Parallel construction of a global queue while avoiding sequential bottlenecks when atomically accessing queue-control variables.

1.3.2 Motivation

Many systems in the real world make use of speech recognition and these systems assist students studying foreign languages with translation; they also assist people with disabilities, online shopping and voice activated passwords and security systems among other uses. Optimizing the performance of automatic speech recognition systems will

help improve the services provided by these systems. Parallelizing the dynamic programming search will help improve the efficiency of the recognition process, hence reducing the latency in processing speech for large vocabulary systems.

1.4. Research Questions (RQ)

These are the questions that the research proposes to answer.

RQ1: How can a speech recognition dynamic programming based search algorithm be developed for GPU platform particularly one that will be compatible and efficient on most recent technological computing platforms?

RQ2: How can it be ensured that the dynamic programming based search algorithm will perform efficiently for parallel architectures especially on the recent trend of high performance computing?

RQ3: How will the performance be evaluated since the then implemented system must have the ability to function on different speech application systems, especially pointing to the current technological advancement in this field?

1.5. Research Goal and Objectives

The goal of this research and the objectives that will be employed to reach the goal are discussed in the following subsections.

1.5.1. Research Goal

The main goal of this research is to optimize the dynamic programming Viterbi search for automatic speech recognition.

1.5.2. Research Objectives

The objectives of this research are:

A. To conduct a study on already existing language corpora, search algorithms and speech recognition systems.

B. To implement a speech recognition dynamic programming based Viterbi search algorithm.

C. To optimise dynamic programming based Viterbi search algorithm on a GPU based platform using CUDA and analyse the performance of the implementations.

1.6. Summary of Dissertation

This section summarizes what will be covered in each chapter.

Chapter 1: Introduction

This chapter presents the general introduction and background, aims and objectives, problem statement, substantiation of study, research methodology, and summary of the dissertation.

Chapter 2: Literature Review

This chapter presents an overview of previous related work and provides a critical analysis of it.

Chapter 3: Methods

Chapter 3 discusses methods that are used in speech recognition research.

Chapter 4: Experimentation

This chapter discusses in detail how the experiments have been conducted, results of the experiments are presented and discussed.

Chapter 5: Summary and Concluding Remarks

This chapter provides a summary of this work and suggests the ideas for future work.

Chapter 2: Literature Review

This chapter presents an overview of previous related work and provides a critical analysis of it.

2.1Speech Recognition on GPUS and using CUDA

2.1.1 Implementing Speech Recognition on GPUs

Poli *et al.* in [**39**] showed the capability of the GPU to perform general purpose computation by developing a speech recognition application inside. They applied Dynamic Time Warping (DTW) on voice password identification. Their experiments achieved a performance improvement of approximately 75%, using GPU against CPU on the first two kernels. Yet in the last kernel which computes the path, processing time between the GPU and the CPU was basically the same with the GPU 5% faster than the CPU.

Yi and Talakoub in [**10**] implemented a speech recognition system on a GPU using CUDA. The speech recognition system was initially developed in MATLAB and it was later converted into a C++ program. Then the C++ version was used as a reference to create the CUDA based implementation. The GPU and CPU implementation performance results revealed a remarkable improvement due to moving the computational expensive tasks onto the GPU. The average execution runtime for 45 iterations on the CPU was 8.3044 seconds and 1.4280 seconds for the GPU. These results translate into a performance improvement of approximately 5.8.

2.1.2 Acoustic computations on GPUs

Some researchers investigated acoustic computations on parallel architectures. Cardinal *et al.* in [**40**] explored how the acoustic likelihood computations can be implemented on a GPU. They implemented the acoustic computation module in CUDA and also studied the speed of a large vocabulary speech recognition system. Their implementation revealed that the GPU is 5× faster than the CPU SSE-based implementation. The enhancement led to a speed up of 35% on a large vocabulary task.

Vesely *et al.* in [**41**] introduced the acceleration of acoustic likelihood computations for graphical processing units. According to their optimal method, they achieved an 11.6× speedup on the acoustic probability evaluation.

Dixon *et al.* in [**42**] used weighted finite state transducer (WFST) based decoding engine that utilized a commodity graphics processing unit (GPU) to perform the acoustic computations to move this burden off the main processor. They described their new GPU scheme that could achieve a very substantial improvement in recognition speed whilst incurring no reduction in recognition accuracy. They evaluated the GPU technique on a large vocabulary spontaneous speech recognition task using a set of acoustic models with varying complexity and the results consistently showed that by using the GPU it is possible to reduce the recognition time, with the largest improvements occurring in systems with large numbers of Gaussians. For the systems which achieve the best accuracy they obtained between 2.5× and 3× speed-ups. The faster decoding times translate to reductions in space, power and hardware costs by only requiring standard hardware that is already widely installed.

2.1.3 Optimising the decoding process using the GPU

Researchers such as Cardinal *et al.* in [**43**] worked on optimizing decoders. They explored how the performance of speech recognition systems can be improved by the use of A* algorithm over the Viterbi algorithm combined with a GPU for the acoustic computations in large vocabulary applications. When compared to the classical Viterbi decoder, the experiments resulted in approximately 8.7× less states being explored. The multi-thread implementation of the A* decoder combined with GPU led to a speed-up factor of 5.2 over its sequential counterpart and an improvement of 5% absolute of the accuracy over the Viterbi search at real-time.

Rehman *et al.* in [**44**] implemented a dynamic programming algorithm (Viterbi) on NVidia graphics processing unit using CUDA and concluded that it was accelerated from 3 to 6 × as compared to the serial execution on central processing unit.

2.1.4 Parallel scalability, neural networks and Gaussian mixture models on GPUs

Researchers such as Liu *et al.* in [**45**] explored neural networks by implementing a parallel Artificial Neural Network (ANN) training procedure, based on block mode back propagation learning algorithm using two different approaches. The first is data parallelization using Portable Operating System Interface (POSIX) threads. The second is node parallelization using GPU with CUDA. They compared the speed-up of both approaches by learning typically-sized network on the real-world phoneme-state

classification task, showing nearly 10× reduction when using the second approach, while the first approach gives only 4× reduction.

You *et al.* in [**46**] investigated parallel scalability in speech recognition. They explored a design space for parallel scalability for an inference engine in large vocabulary continuous speech recognition (LVCSR). Their implementation of the inference engine involves a parallel graph traversal through an irregular graph-based knowledge network with millions of states and arcs. The major challenges were to define a software architecture that exposes sufficient fine-grained application concurrency, to efficiently synchronize between an increasing number of concurrent tasks, and to effectively utilize parallelism opportunities in today's highly parallel processors.

They proposed four application-level implementation alternatives called algorithm styles and constructed highly optimized implementations on two parallel platforms: an Intel Core i7 multicore processor and a NVIDIA GTX280 many core processor. The highest performing algorithm style varied with the implementation platform. On a 44 minutes speech data set, they demonstrated substantial speedups of 3.43× on Core i7 and 10.53× on GTX280 compared to a highly optimized sequential implementation on Core i7 without sacrificing accuracy. The parallel implementations contained less than 2.5% sequential overhead, promising scalability and significant potential for further speed- up on future platforms.

Some studies were also carried out involving Gaussian Mixture Models (GMM). Gupta and Owens in [**47**] optimized compute and memory-bandwidth-intensive GMM computations for low end, small-form-factor devices running on GPU-like parallel processors. They proposed modifications to three well-known GMM computation reduction techniques. They were able to achieve compute and memory bandwidth savings of over 60% and 90% on a 1,000-word, command-and-control, continuous speech task respectively, with some degradation in accuracy, when compared to existing GPU-based fast GMM computation techniques.

2.2 Optimizing decoders

Since decoding has always been a computationally expensive process, some researchers focused on optimising decoding. Hannani and Hain in [**48**] investigated automatic optimization of decoder parameters. They presented an effective and straightforward

approach to decoder parameter optimization based on tracking a curve of best performance for any possible real-time factor. The objective was to find the optimal configuration that yields minimal search errors for any real-time factor. Experiments, conducted using the large vocabulary speech decoder HDecode from the Hidden Markov Model Toolkit, show on a large test set of conversation telephone speech that with modest computational cost optimal performance curves for specific decoders and data types can be obtained. Careful selection of the cost function allows a further reduction of computational cost by 55%.

Kalinli and Naranayan in [**49**] presented an attention shift decoding (ASD) method inspired by human speech recognition. ASD decodes speech inconsecutively using reliability criteria; the unreliable speech regions are decoded with the evidence of reliable speech regions which is contrary to the traditional automatic speech recognition (ASR) systems. On the BU Radio News Corpus, ASD provides significant improvement (2.9% absolute) over the baseline ASR results when it is used with oracle island-gap information. They proposed a new feature set for automatic island-gap detection which achieves 83.7% accuracy. They also proposed a new ASD algorithm using soft decision to cope with the imperfect nature of the island-gap classification. The ASD with soft decision provides 0.4% absolute (2.2% relative) improvement over the baseline ASR results when it is used with automatically detected islands and gaps.

Stoimenov and Schultz in [**50**] explored the benefits of decoding with an optimized speech recognition network over the fully task-optimized prefix-tree based decoder IBIS. They designed and implemented a new decoder called SWIFT (Speedy Weighted Finite-state Transducer) based on WFSTs with its application to embedded platforms in mind. They presented evaluation results on a small task suitable for embedded applications, and on a large task, namely the European Parliament Plenary Sessions (EPPS) task from the TC-STAR project. The SWIFT Decoder is up to 50% faster than IBIS on both tasks. In addition, SWIFT achieves significant memory consumption reductions obtained by their innovative network specific storage layout optimization.

2.3 Dynamic programming algorithms

Hachkar et al. in [**51**] used two algorithms to implement a system of Automatic Recognition of isolated Arabic Digits: Dynamic Time Warping (DTW) and Discrete Hidden Markov Model (DHMM). The endpoint detection, framing, normalization, Mel Frequency Cepstral Coefficient (MFCC) and vector quantization techniques were used to process speech

samples to accomplish the recognition. DTW-based system recognition led to recognition accuracy of 77%. The better recognition accuracy of about 92% was obtained with DHMM-based system. The recognition performances for the two ASR systems were worse in noisy environment, but the pattern recognition using HMM was better than the pattern using DTW.

Amin and Mahmood in [**52**] explored speech recognition using dynamic time warping. In their research, they described an isolated word, speaker dependent speech recognition system capable of recognizing spoken words at sufficiently high accuracy. They showed increased memory efficiency offered by using speech detection for separating the words from silence, and improved system performance achieved by using Dynamic Time Warping, while keeping in view the overall design process. Supported by experimental results, the system was tested and verified on MATLAB as well as the TMS320 C6713 DSK with an overall accuracy exceeding 90%.

Lipeika *et al.* in [**32**] used Vector quantization to create reference templates for speaker recognition when they developed an isolated word speech recognition system based on dynamic time warping (DTW) and linear predictive coding features (LPC). They evaluated performance using 12 words of Lithuanian language pronounced ten times by ten speakers. The results of their experiments showed that recognition error rate in speaker dependent mode were 0.83% and 1.94% in speaker independent mode. In speaker independent mode, using vector quantization; they obtained the best results when vector quantization was based on splitting a cluster with largest average distortions into two clusters. Recognition error rate increased to 2.5%. This slight increase in error rate reduced Computation amount significantly.

2.4 Optimising Dynamic programming algorithms

Wei and Weisheng in [**53**] attempted to improve recognition efficiency without compromising recognition accuracy. They analysed the traditional Viterbi-Beam search algorithm and proposed an improved adaptive Viterbi- Beam search algorithm by analysing the voice activity model of different stages. The method of combining Viterbi algorithm with Beam pruning technique is useful to compress the search space, which reduces the computational complexity. The experimental results showed that the search space was compressed effectively without affecting recognition accuracy and an improvement on search efficiency of 35.77% was observed.

Ortmanns and Ney in [54] presented the time-conditioned approach in dynamic programming search for large-vocabulary continuous speech recognition. Their approach has been successfully tested on the NAB task using a vocabulary of 64 000 words.

2.5 Optimising Feature extraction

Li and Lee in [55] proposed a discriminative learning framework, called soft margin feature extraction (SMFE), for jointly optimizing the parameters of transformation matrix for feature extraction and of hidden Markov models for acoustic modelling. SMFE was tested on the TIDIGITS connected digit recognition task; the proposed approach achieved a string accuracy of 99.61%, much better than their previously reported soft margin estimation (SME) framework results.

Pour and Farokhi in [56] presented an advanced method that developed an automatic Persian speech recognition system performance. In their method, the recorded signal is pre-processed so that its section includes reducing the noise with Mels Frequency Cepstral Analysis and feature extraction using discrete wavelet transforms (DWT) coefficients; then the extracted features are fed to Multilayer Perceptron (MLP) network for classification. According to the results their method was able to classify speech signals using UTA algorithm redounded to increase system learning time from 18000 to 6500 epoch and system accuracy average value to 98% at the minimum time.

2.6 Improving speech recognition performance

Juang *et al.* in [57] proved that the Minimal Classification Error (MCE) approach achieves better performance than the traditional probability distribution estimation approach in a number of speech recognition experiments. The MCE method generally provides 30–50% reduction in error rate, compared to the traditional recognizer design.

Guezouri *et al.* in [58] developed an approach based on Temporal Radial Basis Function (TRBF) which had many advantages such as few parameters, speed convergence and time invariance. Their application aimed to identify vowels taken from natural speech samples from the Timit corpus of American speech. They reported a recognition accuracy of 98.06% in training and 90.13% in tests on a subset of 6 vowel phonemes, with the possibility to expand the vowel sets in future.

Lecouteux *et al.* in [59] proposed an integrated approach where outputs of secondary systems are integrated in the search algorithm of a primary one. DDA was evaluated on a

subset of the ESTER I corpus consisting of 4 hours of French radio broadcast news. Results demonstrated that DDA significantly outperforms vote-based approaches: they obtained an improvement of 14.5% relative word error rate over the best single-systems, as opposed to the 6.7% with a ROVER combination. An in-depth analysis of the DDA showed its ability to improve robustness and a relatively low dependency on the search algorithm. The application of DDA to both A* and beam-search-based decoder yielded similar performances.

2.7 Critical Analysis

Researchers, working on the very promising and challenging field of automatic speech recognition, are collectively heading towards achieving natural conversation between Human beings and machines, are applying the knowledge from areas of Linguistics, Artificial Intelligence, Neural Networks, Acoustic-Phonetics, Speech Perception among other techniques. The challenges to the recognition performance of ASR are being provided concrete solutions so that the gap between recognition capability of machine and that of a human being can be reduced to a maximum extent according to literature some of the methods used solved one problem and created another, looking at Lipeika et al. [32], their work yielded Word Error rate increased and it lead to significant reduction of computation amount.

Chapter 3: Methods

3.1 Speech Recognition Performance

The performance of a speech recognition system is usually assessed by means of accuracy and speed. Accuracy is measured in terms of Word Accuracy Rate (WAR), Word Error Rate (WER), Single Word Error Rate (SWER) and Command Success Rate (CSR). Speed is measured according to the Real Time Factor (RTF). The Word Error Rate and the Real Time Factor are the most common metrics for measuring the performance of a speech recognition system.

3.1.1 Real Time Factor (RTF)

The RTF is derived from the ratio of the time taken to process an input over the actual duration (length) of that particular input. The equation is as follows:

$$RTF = \frac{P}{I_D} \quad \text{..... (1)}$$

Where P= Processing time and I_D = Length of input. The processing is done in real time only if the value of the real time factor is equal to 1 or less than 1.

3.1.2 Word Error Rate (WER) and Single Word Error Rate (SWER)

The WER is the common metric for speech recognition performance. It is derived from the Levenshtein distance, working at the word level (sentence context based algorithm) instead of the phoneme level. Conversely the SWER is used on raw words from the phonemes. The WER equation is derived from the ratio of word insertion, substitution and deletion errors in a transcription to the total number of uttered words.

$$WER = \frac{S+D+I}{N} \quad \text{..... (2)}$$

OR

$$WER = \frac{S+D+I}{S+D+C} \quad \text{..... (3)}$$

Where S= number of substitutions,

D= number of deletions.

I = number of insertions.

C = number of correct words.

N = number of words in the reference.

Thus N = S + D + C.

WER is a valuable tool for comparing different systems as well as evaluating improvements within one system [**60**].

3.1.3 Word Accuracy Rate (WAR)

WAR is another metric for system performance; accuracy is based on the edit distance of the speech recognition transcription and the reference transcription.

$$\text{WAR} = 1 - \text{WER}$$

$$= \frac{N-S-D-I}{N}$$

$$= \frac{H-I}{N} \quad \text{..... (4)}$$

Where H = N – (S+ D).

3.1.4 Command Success Rate (CSR)

CSR is usually used in dialogue systems, whereby the dialogue engine and the task help guide the speech recognition. It is derived from the ratio of successful commands to the total number of commands issued.

$$\text{CSR} = \frac{S_C}{I_C} \quad \text{..... (5)}$$

Where S_C = successful commands

I_C = Issued commands.

3.2 Types of Recognition

Speech recognition systems can be classified by describing the types of utterances they have the ability to recognize. The classes are presented in subsection 2.4.1 to 2.4.4.

3.2.1 Isolated Words

Isolated word (isolated utterance) recognizers usually require each utterance to have absence of audio signal on both sides of the sample window. It accepts single words or

single utterances at a time. These systems have "Listen/Not-Listen" states, where they require the speaker to wait between utterances [**61**].

3.2.2 Connected Words

Connected word systems or more correctly 'connected utterances' are similar to isolated words, but allow separate utterances to be 'run-together' with a minimal pause between them. [**34**]

3.2.3 Continuous Speech

Continuous speech is basically computer dictation. Continuous speech recognizers allow users to speak almost naturally, while the computer determines the content. It is very difficult to create recognizers with continuous speech capabilities because they utilize special methods to determine utterance boundaries [**61**].

3.2.4 Spontaneous Speech

At an elementary level, this can be thought of as speech that is natural sounding and not rehearsed. An ASR system with spontaneous speech ability should be able to handle a variety of natural speech features such as words being run together, "ums" and "ahs", and even slight stutters [**34**].

3.3 Search

Searching is a method that can be used by a computer to move around, examining a problem space, and make decisions about whether the goal (sought object) has yet been found. There are two methods of searching which roughly correspond to top-down and bottom-up approaches: data-driven and goal-driven search. According to Coppin in [**62**] data-driven search, also known as forward chaining, starts from an initial state and uses actions that are permissible to move forward until a goal is reached.

Goal-driven search, also known as backward chaining is a search which starts at the goal and work back toward a start state, by observing what moves could have led to the goal state. Both approaches end up producing the same results, but depending on the nature of the problem being solved, in some cases one can run more efficiently than the other [**62**].

Search can either be Uninformed (blind) or Informed (heuristic). Uninformed search include Depth-First Search and Breadth-First Search while Informed search include A* search (Best-First Search).

The general difficulty with the search process lies in the fact that the output can have a length varying from the reference word sequence. This problem is solved by aligning the recognised word sequence with the reference word sequence using dynamic string alignment. The equations 7 and 10 are used to find the most likely string of words $\hat{w}$ given the data in a HMM-based Bayesian recognizer.

$$\hat{W} = \underset{W}{argmax}\, P(w|x) \quad \text{..... (6)}$$

$$= \underset{W}{argmax}\, F(x|w)\, P(w) \quad \text{..... (7)}$$

P(w) is a probability estimate given by the language model. $F(x|w)$ is the probability sequence of acoustic observations conditioned by a given word sequence W and is estimated by the acoustic model [**12**] .

$$\underset{W}{argmax}\, P(w|x) = \underset{W}{argmax}\, \frac{F(x|w)\, P(w)}{F(x)} \quad \text{..... (8)}$$

$$= \underset{W}{argmax}\, F(x|w)\, P(w)$$

$$= \underset{W}{argmax} \sum_{q} P(x|q,w)\, P(q|w)\, P(w) \quad \text{..... (9)}$$

$$= \underset{W}{argmax} \sum_{q} P(x|q)\, P(q|w)\, P(w) \quad \text{..... (10)}$$

Figure 2.2 shows the operations of the search process during speech recognition.

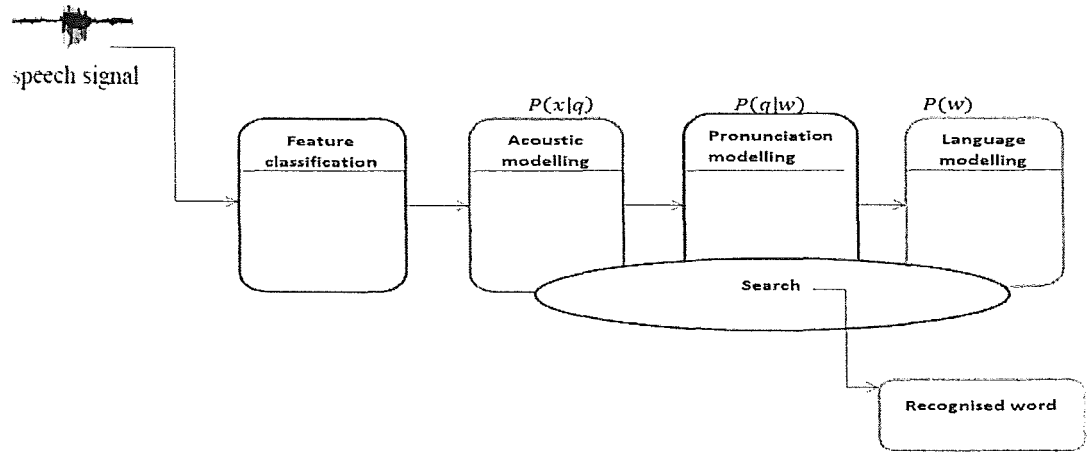

Figure 3.1: Probability estimates during the search process

3.4 Template Based and Model Based Speech Recognition Approaches

The following subsections discuss the template based approach and the model based approach.

3.4.1 Template Based Approach

The Template based approach has no statistical training. It directly aligns the testing and reference waveforms on their feature vector sequences to derive the overall distortion between them [**63**]. The decision rule is used to choose the reference pattern (R^*) with the smallest alignment distortion D(T, R^*).

$$R^* = \underset{v}{argmin}\ D(T, R_V) \quad \text{..... (11)}$$

In this approach Dynamic Time Warping (DTW) is used to compute the best possible alignment warp (Φ_v) between the testing waveform T and the reference waveform (R_V) and the associated distortion D(T, R_V) [**64**]. The Template based approach is fairly effective with small vocabulary isolated word speech recognition due to the fact that DWT is a simple algorithm to implement.

3.4.2 Model Based Approach

The Model based approach joins the sub word models according to the pronunciation of the words in the lexicon. HMMs are used for statistical training in this approach. The model based approach uses a search process to uncover the word sequence $\widehat{W}$ = $w_1, w_2, \ldots, w_m$ that has the maximum posterior probability P(w| x). Consequently the model based continuous speech recognition is both pattern recognition and a search problem. Usually the model based approach uses Viterbi search or A* stack decoders [**63**].

3.5 Dynamic Programming and Dynamic Programming Algorithms

In the following subsections dynamic programming and dynamic programming algorithms are discussed.

3.5.1 Dynamic Programming

Dynamic Programming is an optimization approach that transforms a complex problem into a sequence of simpler problems. Its essential characteristic is the multistage nature of the optimization procedure. The main idea of Dynamic programming is to set up a

recurrence relating the solution of a larger instance to the solution of the smaller instances, solve the small instances and store their results in a table, then finally extract the solution to the larger instance from that table. Dynamic programming has a complexity of O (n^2), which can cause unreasonable demands on both the processing time and system memory.

3.5.2 Dynamic programming algorithms

The dynamic programming algorithm functions by distorting one waveform onto the axis of the other. However, the algorithm attempts to match the waveforms so that the similarities are maintained and time aligned instead of simply stretching or compressing the waveforms. The most common dynamic programming algorithms are discussed below.

3.5.2.1 Viterbi Algorithm

The Viterbi algorithm can be considered as the dynamic programming algorithm applied to the HMM (Hidden Markov Models) or as an altered forward algorithm. It requires memory complexity of O(N T) and computation time complexity of $O(N^2T)$. An HMM is a stochastic technique into which selected temporal information can be integrated. In addition [**56**] stated that Hidden Markov Models are finite automata that are given a number of states; moving from one state to another is made immediately at similarly spaced time instants. At every pass from one state to another, the system generates observations. Two processes are taking place: the transparent one, represented by the observations string (feature sequence), and the hidden one, which cannot be observed, represented by the state string. The Viterbi algorithm shown in Figure 2.3 picks and remembers the best path, instead of summing up probabilities from different paths coming to the same destination state.

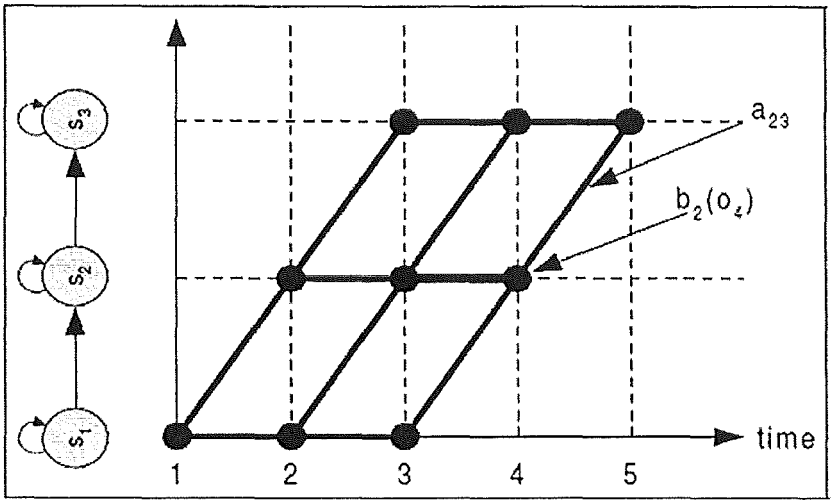

Figure 3.2: Viterbi algorithm

This algorithm is used in most communication devices to decode messages in noisy channels; it also has widespread applications in speech recognition [**43**].

3.5.2.2 Dynamic time warping (DTW) algorithm

The Dynamic Time Warping (DTW) algorithm is also a Dynamic Programming technique based algorithm. However, this algorithm is for recursively measuring similarities between two sequences of feature vectors which may vary in time or speed to lighten distortion. This technique is also used to find the optimal alignment between two time series if one time series may be "warped" non-linearly by stretching or shrinking it along its time axis [**61**]. Dynamic Time Warping creates a similarity matrix for two utterances and uses dynamic programming to find the lowest cost path. This warping between two time series can then be used to compute the best possible alignment warp between the test pattern and the reference pattern as well as the associated distortion.

3.5.2.3 Baum-Welch algorithm (forward-backward algorithm)

The forward-backward or Baum-Welch algorithm is a dynamic programming algorithm, and is closely related to the Viterbi algorithm for decoding with HMMs or CRFs. It derives its name from the fact that, for each state in an execution trellis, it computes the 'forward' probability of arriving at a particular state and the 'backward' probability of generating the final state of the model, given the current approximation. This algorithm estimates the parameters of a Hidden Markov Model (HMM) by Expectation- Maximization (EM), using dynamic programming

to carry out the expectation steps efficiently. Baum-Welch algorithm functions by alternating expectation and maximization steps, it maximises the probability of observed training data and only finds a local maximum, and therefore it is sensitive to initial conditions [**65**]. In speech recognition Baum-Welch algorithm is used in the data training phase.

3.5.2.4 Forward Algorithm

Forward algorithm is used as a tool to evaluate an HMM. It is used to calculate a 'belief state': the probability of a state at a certain time, given the history of evidence.

DTW and Viterbi are time-synchronous searches. They both look like breadth-first with pruning.

3.6 Optimization Techniques

From subsections 2.8.1 and 2.8.2.2 we discuss some optimization techniques that are possible for dynamic programming algorithms.

3.6.1. Loop transformation techniques

Loop transformation is also called loop optimization. It is a process of increasing execution speed and reducing the overheads associated with loops. There are various loop transformation techniques some of which are discussed in subsections 2.8.1.1 to 2.8.1.6. Their significance lies in making use of parallel processing capabilities and improving cache performance.

3.6.1.1. Loop Fusion

Loop fusion [**66**], also known as loop jamming, combines two adjacent isomorphic loops. For two loops to be fused they must both have the same loop bounds, and the statements in the fused loop must not exhibit any backward dependencies. Its benefits are: reducing the loop overhead and improving data reuse and data transfer.

3.6.1.2. Loop Unrolling

Loop unrolling [**66**] is a simple transformation that aggregates successive instances of loop iterations in the body without loop controls and increases the loop step by the same factor. This divides the loop overhead by a factor (f), and increases the

possibilities for common sub-expression. It also promotes reuse since identical and consecutive values appear multiple times in the unrolled loop body. The example below shows a loop before unrolling.

```
do i = 2, n-1
k[i] = k[i] + k[i-1] * k[i+1]
end do
```

The following example shows a loop with an unrolling of factor 2.

```
do i = 1, n-2, 2
k[i] = k[i] + k[i-1] * k[i+1]
k[i+1] = k[i+1] + k[i] * k[i+2]
end do
if (mod(n-2,2) = 1) then
k[n-1] = k[n-1] + k[n-2] *k[n]
end if
```

The upper loop bound must be altered to stay in its original range and a small fix up conditional statement or loop may be needed afterwards to finish the last nmod statements. Since unrolling causes the code to increase it may lead to i cache misses.

3.6.1.3. Loop unroll and jam

Loop unroll and jam operates by unrolling the outer loop and fusing the new couples of the inner loop. It increases the size of the loop body and hence the available instruction level parallelism. Loop unroll and jam also has the possibility of improving the data locality [**67**].

3.6.1.4. Loop interchange

Loop interchange operates by exchanging the position of two loops in a loop nest. This can improve the time of execution by one or two orders of magnitude. For example, by moving a parallel loop outwards, the necessarily serial work is moved towards the inner loop, increasing the amount of work done per fork-join operation.

It is used to improve cache behaviour and can be used to control the granularity of the work in nested loops [**67**]and [**66**] .

3.6.1.5. Loop blocking

Loop blocking breaks the entire loop into chunks. This is mainly done on the iteration space and can be seen as task partitioning [**67**]. It derives a coarse grained parallelism from a fine grained model, improves cache performance and handles memory constraints.

3.6.1.6. Scalar Replacement

Scalar replacement simply deals with the frequent reuse of a fixed array element in an inner loop. In the following code, total[g] is repeatedly read and written in the inner loop [**66**]. The examples below show the loop before and after the application of scalar replacement.

```
do g = 1, n

do h= 1, n

total[g] = total[g] + a[g,h]

end do

end do
```

after applying scalar replacement the fixed array element can be assigned to a scalar before the inner loop, and if modified, stored back into the original element afterwards. This saves index calculations and reduces the total number of accesses to the array element thus reducing memory traffic.

```
do g= 1, k

T = total[g]

do g = 1, k

S = S + a[g,h]

end do

total[g] = S
```

end do

3.6.2. Load Balancing

Load balancing [62] is a technique for distributing workloads across multiple resources and this can be done by work sharing or work stealing. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid the overload of any one of the resources. It can either be dynamic or static.

3.6.2.1 Dynamic Load Balancing Algorithms

Dynamic load balancing algorithms attempt to utilize the runtime state information to make more informative load balancing decisions. Indisputably, the static approach is easier to implement and has minimal runtime overhead. Nonetheless, dynamic approaches may produce better performance [62].

3.6.2.2 Static Load Balancing Algorithms

Static load balancing algorithms assume that all information governing load-balancing decisions that can include the characteristics of the jobs, the computing nodes, and the communication network are known in advance. Load-balancing decisions are made deterministically or probabilistically at compile time and remain constant during runtime. Static algorithms have one major disadvantage in that they assume that the characteristics of the computing resources and communication network are all known in advance and remain constant [62].

3.7 GPUs

The GPUs (Graphics Processing Units) are the processors on the graphics cards. These processors are enhanced for 2D or 3D graphics, video, visual computing, and display. GPUs are highly parallel, highly programmable and highly multithreaded multiprocessors optimized for visual computing. They offer real-time visual interactions with computed entities through graphics images, and video. They serve as both a programmable graphics processor and a scalable parallel computing platform.

The GPU has already passed the speed of the CPU, and is far ahead. GPUs have become an attractive solution for High Performance Computing (HPC) in terms of performance and acquisition cost. GPUs have evolved into a very attractive hardware platform for general purpose computations due to their extremely high floating-point processing performance, huge memory bandwidth and their comparatively low cost [68]. The rapid evolution of

GPUs in performance, architecture, and programmability can provide application potential beyond their primary purpose of graphics processing. The GPU architecture is shown in Figure 3.3.

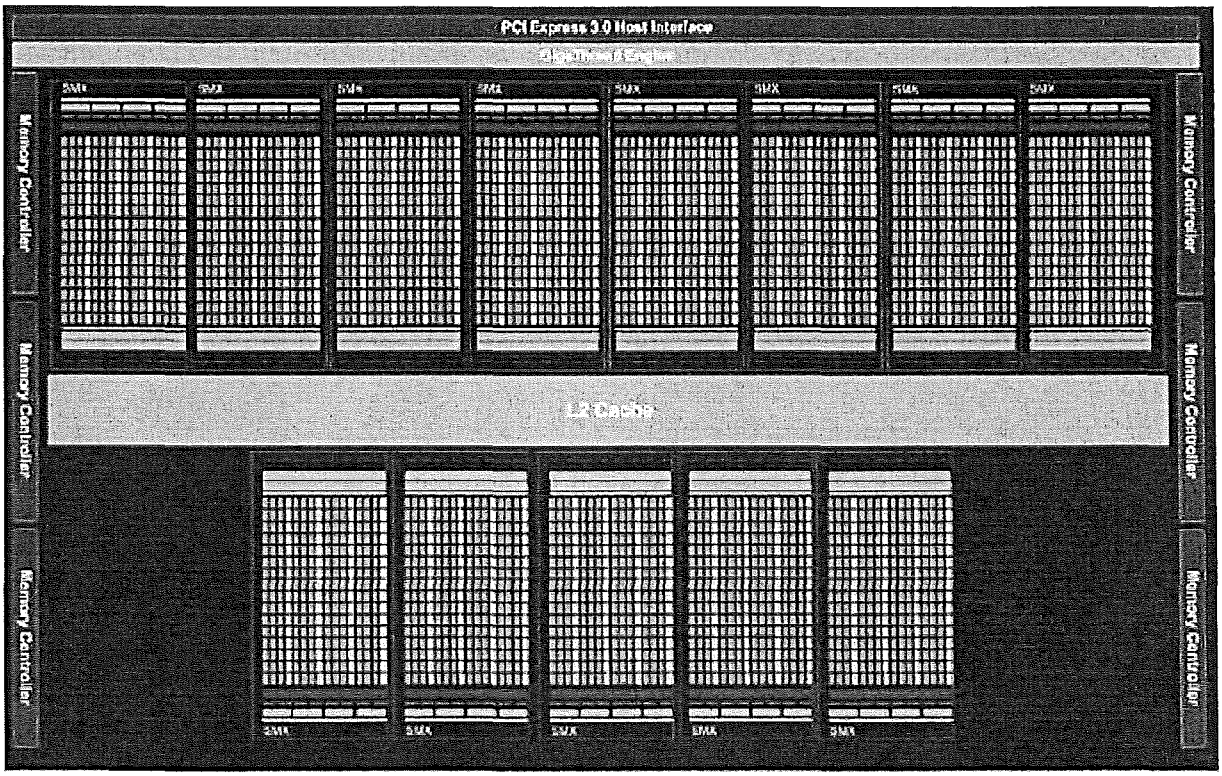

Figure 3.3: GPU (Tesla) architecture adapted from [69]

Modern NVIDIA GPUs are throughput-oriented many core processors that offer very high peak computational throughput [**70**]. In that way they can be programmed to render not only geometry, but also the solution to other, more general problems. Such programming of the GPU is often called GPGPU (General Purpose computation on Graphics Processing Units), and has become a field of great interest. The reason is its vast processing capabilities.

GPUs achieve a yearly growth in terms of flops that have significantly outnumbered Moore's law which states that the number of transistors per chip doubles roughly every second year, and the performance is doubled around every 18th month.

In recent years, these devices have become increasingly programmable and therefore useful for general purpose workloads. The HPC community in particular has invested many resources in improving the performance of computational programs on GPU architectures. GPU devices are commonly programmed using low-level programming models, of which

the two most common are CUDA and OpenCL. In both models, the programmer writes an imperative program (called a kernel) that is executed by each thread on the device [71].

3.8 CUDA

CUDA – Compute Unified Device Architecture is a C-based programming model for GPUs. It was introduced together with GeForce 8800 with Joint CPU/GPU execution (host/device). CUDA is the comprehensive software and hardware architecture for GPGPU that was developed and released by NVidia in 2006. CUDA includes C/C++ Software development tools, functions libraries and a hardware abstraction tool that hides the GPU hardware from developers [72]. The major design goal of CUDA is to support heterogeneous computations in a sense that serial parts of an application are executed on the CPU and parallel parts on the GPU. Figure 3.4 represents the CUDA application architecture.

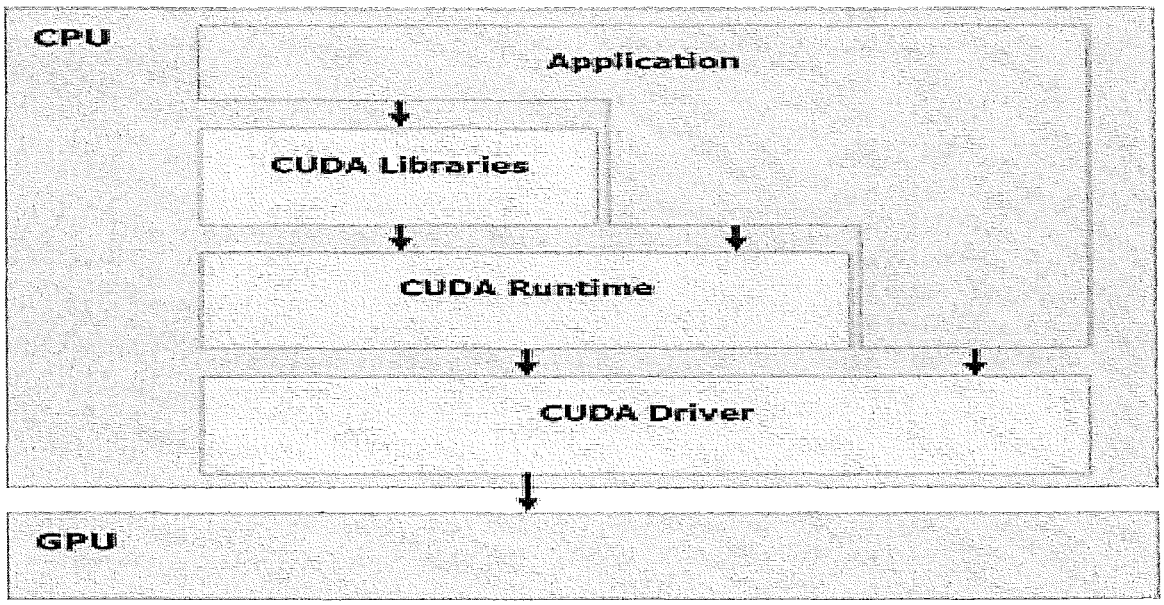

Figure 3.4: NVidia CUDA application architecture adapted from [73]

CUDA programming model is highly scalable and relatively easy. It requires programmers to write special code for parallel processing but it does not require them to unambiguously manage threads, which simplifies the programming model. The automatic thread management reduces the parallel programming complexity significantly. It also allows for explicitly defining parallelism of selected parts of the code. In CUDA, a group of threads work together in round-robin fashion, ensuring that each thread gets execution time without

delaying other threads, thereby reducing the thread overheads. The wait for remote access and service strongly factors into a CUDA's efficiency and scaling.

Chapter 4: Experimentation

This chapter provides a brief description on how the actual research experiments were conducted. It also specifies a list of experimental tools used and the results of the implementations and results analysis and evaluation of the implementations.

4.1. Introduction

Recent work in the area of automatic speech recognition has brought about a lot of developments. Today's Automatic Speech Recognition systems are much improved in speech processing. However, to develop these systems requires a lot of effort, since there are a lot of factors such as the developmental environment, noise, and system components capabilities to be considered when building them. The actual speech processing by a system is also a computationally demanding task since there are many processes to go through during these computations. Thus, to achieve better speech results optimum speech processing enhancements must be deployed. During the speech recognition process, the system goes through a lot of decisive system applications that best form the actual processing capability of the overall system.

The output of an ASR is actually the result of a very rigorous system search which involves the act of matching the recognized output string with the initial acoustic input string. This search is actually a parallel process since it is performed in very irregular graph structures. This is a challenge since this parallelism nature of searching requires powerful processing capable devices to better enhance the actual search process. As an effort to achieve better speech results, this work is implemented on a CUDA GPU based platform by optimizing the dynamic programming search of the automatic speech recognition system.

4.2. Description of the Research Methodology

This work followed three processes which formed the overall research methodology. The first being the dynamic programming search method survey, which was done to look at what recent speech processing improvements have been achieved using this approach particularly on CUDA GPUs. Secondly was the experimental setup, which focused on the actual experimental tools and how they were setup. Thirdly as proof of concept, performance comparisons between the CUDA GPU optimized dynamic programming Viterbi search algorithm, dynamic programming Viterbi search algorithm with loop unrolling and the

original algorithm were done. Thorough analysis and discussions are then done with regard to the performances of the three algorithms.

4.3. Experimental System Tools and Platforms

HTK was installed on both the Linux based workstation with Intel core *i*7-3770 CPU and the GPU based system running on GTX 8800. We use two systems because we want to observe whether the algorithm performs better on the CPU or on the GPU. Sound data from the reference files as well as the test files was loaded and a speech recognition Viterbi search algorithm was implemented on both systems. The Viterbi search algorithm was optimised by loop unrolling to improve the optimality of the search process and thus improving the efficiency of the recognition process. CUDA was used to implement the optimised version of the speech recognition Viterbi search algorithm on the GPU based system using the already existing language corpus. Performance results of the implementations were also thoroughly analysed in section 5.3.2 and evaluated in section 4.9. Tools used include the following:

A. HTK version 3.4.1
B. CMU Sphinx 4
C. GPU GTX 8800
D. NVidia CUDA 5.5 Toolkit
E. Linux based workstation with Intel core i7-3370 CPU only
F. CMU US BDL Arctic 0, 95 Speech corpus

4.3.1. Description of tools

Subsections 4.3.1.1 to 4.3.1.6 describe the tools that were used in our experiments.

4.3.1.1. The HTK

HTK is a toolkit for building and manipulating HMMs. it is mainly used for speech recognition research although it has been used for various other applications including research into speech synthesis, character recognition and DNA sequencing. HTK consists of a set of library modules and tools available in c source form. The tools provide sophisticated abilities for speech analysis, HMM training and testing and result analysis. The software supports HMMs using both continuous density mixture Gaussians and discrete distributions and can be used to build complex HMM systems.

4.3.1.2. The GeForce GTX 8800

The **GeForce GTX 8800** GPU implements a massively parallel, unified shader design consisting of 128 individual stream processors running at 1.35 GHz. Each processor is capable of being dynamically allocated to vertex, pixel, geometry, or physics operations for the utmost efficiency in GPU resource allocation and maximum flexibility in load balancing shader programs. Efficient power utilization and management delivers industry-leading performance per watt and performance per square millimetre. This core was used in this work due to its powerful computational speed and power in terms of data computation. This core offers great digital processing capabilities other than the GeForce 8800 GTS and the GeForce 8800 GT of its family product. Table 4.1 provides some specifications of NVidia GeForce GTX 8800 GPU.

Table 4.1: Essential specifications of the NVIDIA GeForce GTX 8800.

CUDA Cores	575
Texture Fill Rate (billion/sec)	36.8
Memory Clock	900 MHz
Memory Interface Width	384-bit
Memory Bandwidth (GB/sec)	86.4

4.3.1.3. The CPU Core (Intel core i7-3370)

Intel core i7-3370 is part of the fourth generation Intel core family. Intel core i7-3770 CPU was used in this research work due to its enhanced processing speed and throughput computational power. It has features such as Turbo boost technology, hyper-threading technology, built-in visuals, integrated memory controller, smart cache, virtualisation technology, advanced encryption standards, new instructions and thermal solution for boxed processors. Table 4.2 provides the specifications of Intel core *i*7-3770 CPU.

Table 4.2: Essential specifications of Intel core i7-3770 CPU.

Number of Cores	4
Number of Threads	8
Clock Speed	3.4 GHz
Maximum Turbo Frequency	3.9 GHz
Cache	8 MB
Number of Memory Channels	2
Maximum Memory Size	32 GB
Maximum Memory Bandwidth	25.6 GB/s

4.3.1.4. CUDA Version

CUDA 5.5 is the latest CUDA platform and programming model. It has features such as guided performance analysis, ARM support and it is also optimized for MPI applications. The CUDA development environment relies on tight integration with the host development environment, including the host compiler and C runtime libraries, and is therefore only supported on distribution versions that have been qualified for the CUDA Toolkit release. The CUDA 5.5 Toolkit adds support for Linux on the ARMv7 Architecture. The toolkit comes with a comprehensive set of tools to develop applications for Linux on ARMv7, either natively or cross-platform.

4.3.1.5. Linux Environment

Linux Ubuntu is built on the foundation of Linux, which is a member of the UNIX family. The Linux kernel is best described as the core (almost the brain) of the Ubuntu operating system. The Linux kernel is the operating system's controller; it is responsible for allocating memory and processor time. It can also be thought of as the program which manages any and all applications on the computer itself. Linux environment was used in this work mainly for HTK support. The Linux version used (12.04 LST (Long Term Support)) was chosen for its stable status.

4.3.1.6. The Speech Corpus

Speech corpus is a large collection of audio recordings (sound data) of spoken language. Most speech corpora also have additional text files (reference files) containing transcriptions of the words spoken and the time each word occurred in the recording. We used CMU US BDL Arctic 0, 95 speech corpus which is an English database that is recorded under studio conditions, packaged with information such as phonetic labels and pitch mark files. The corpus contains 1132 utterances spoken by a male speaker, 10045 words and 39153 phones. CMU US BLD Arctic was used in this work due to limited resources of creating own speech corpus. It is also easily accessible than other speech databases.

4.4. System training

HTK integrated CMU sphinx 4 was used as a speech recognition tool. For the system training stage forward-backward (Baum-Welch) algorithm was used. We used CMU US BDL Arctic 0, 95 speech corpus containing 1132 utterances spoken by a male speaker for the training stage. The Hidden Markov Models were connected together in a sequence to enable continuous speech.

4.5. CPU Implementation of the Viterbi Search Algorithm

CPU implementation of the Viterbi search algorithm was performed using C^{++}. The system was developed using HTK. MFCC was used as a feature extraction technique due to its simplicity. Figure 4.1 depicts how the system operates in the CPU platform.

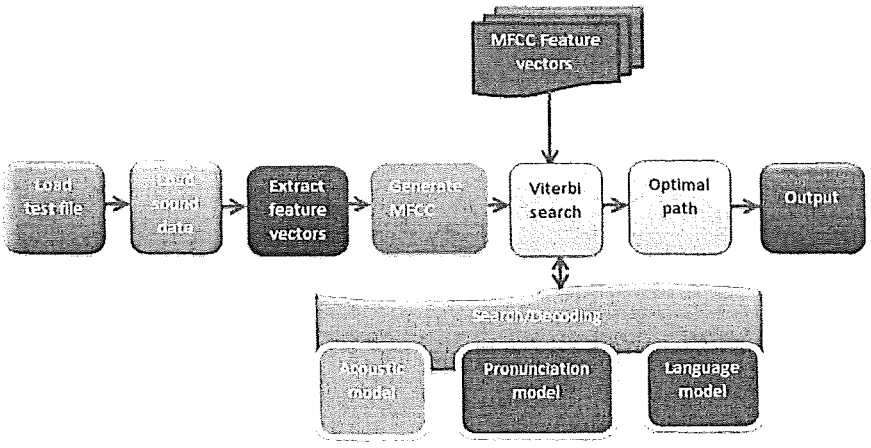

Figure 4.1: CPU implementation of Viterbi Search Algorithm

Firstly the sound data were loaded from the reference files, and then the test files followed. We found it more efficient to use pre-recorded sound to test the system. The feature vectors were extracted during the windowing and segmentation stage as depicted in Figure 4.1. These feature vectors were used to generate MFCC feature vectors for each of the loaded files. The MFCC feature vectors together with the number of states formed a matrix that was used to find the optimal path to the most probable word sequence that matches with the input. The implementation was optimised by the application of loop unrolling on the Viterbi search algorithm. The performance of the optimised implementation yielded considerably good results considering the fact that it performed more efficient than the original Viterbi implementation.

4.5.1 Viterbi Search Algorithm

Viterbi search performs its task in four steps which are initialisation, recursion, termination and backtracking.

Initialisation

$\delta_1(i) = \pi_i b_j(o_1) \qquad 1 \leq i \leq N$

$\Psi_1(i) = 0$

Where: δ_1 is the likelihood score of the optimal sequence of hidden states at initialization (i.e. $t = 1$).

π_i is the initial state probability, b_j is the emission probability.

o_1 is the initial state observation, Ψ_1 is the initial state path.

Recursion

$$\delta_i(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_{ij} \qquad 2 \leq t \leq T, \ 1 \leq j \leq N$$

$$\Psi_t = \operatorname{argmax}_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] \qquad 2 \leq t \leq T, \ 1 \leq j \leq N$$

Where:

a_{ij} is the state transition probability.

Termination

$$P^* = \max_{1 \le i \le N} [\delta_T(i)] \qquad qT^* = \operatorname{argmax}_{1 \le i \le N} [\delta_T(i)]$$

Backtracking

$$qt^* = \Psi_{t+1}(qt^*_{t+1}); \qquad t = T-1, T-2, \ldots, 1$$

Where:

qt* is latent Markovian state.

4.5.1.1. Parallelizing Viterbi Algorithm

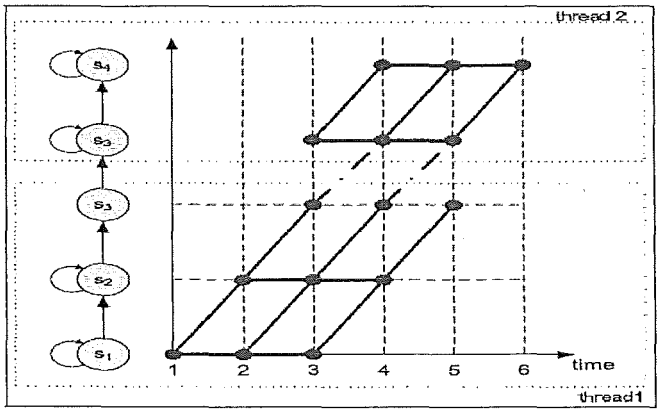

Figure 4.2: Parallel Viterbi algorithm

At each frame, the set of active states is divided into subsets and distributed among the available cores dedicated to the state expansion process (see Figure 4.2). Note that some states have transitions coming from states belonging to different cores. Updating these states simultaneously creates a race condition and can lead to data incoherency. We avoided this problem by duplicating state information and merging them after all states had been expanded.

4.5.2 Optimizing Techniques

Sections 4.5.2.1 and 4.5.2.2 discuss the optimization techniques which were applied in the experiments namely loop unrolling and barrier synchronisation for ensuring thread cooperation on the GPU implementation.

4.5.2.1 Barrier Synchronization

The barrier synchronization primitive is a function call in CUDA. This primitive does not allow a thread execution to move on unless all threads in its block have executed _syncthreads ( ). This primitive operates as shown in the example below

```
for (unsigned int step = 1;step <= blockDim.x; step *= 2)

{
__syncthreads ( );
if (t % step == 0)
PartialSum [2*t] + =PartialSum [2*t+ step];
}
```

4.5.2.2 Loop unrolling

Loop unrolling was done by dividing the loop overhead by a factor of three. Identical values appear multiple times in the loop body. For example if we have a loop like this:

```
(i=1; i<n; i++)

  u[i] = s[i]/u[i-1] ;
```

When we unroll it by a factor of three the code becomes:

```
for ( i = 1; i<=n - 3; i+=3 ) {
   u[i] = s[i] / u[i-1];
   u[i+1] = s[i+1] / u[i];
   u[i+2] = s[i+2] / u[i+1];
      u[i+3] = s[i+3] / u[i+2];
   }
```

4.6. GPU Implementation of the Viterbi Search Algorithm Using CUDA

The optimised Viterbi search algorithm implementation was used to code the CUDA based GPU implementation. In our implementation we took advantage of the GPU's shared memory because it is faster than global memory and also to reduce the load on the global memory. We made use of the shared memory only for communication intensive processes due to the memory limitations associated with shared memory. For this implementation we synchronised the threads to ensure that the parallel threads cooperate in order to yield correct results and avoid deadlocks. We implemented a barrier synchronisation primitive

_syncthreads ( ) which is provided by CUDA. This implementation performed efficiently. Figure 4.3 shows how the parallelised implementation operates.

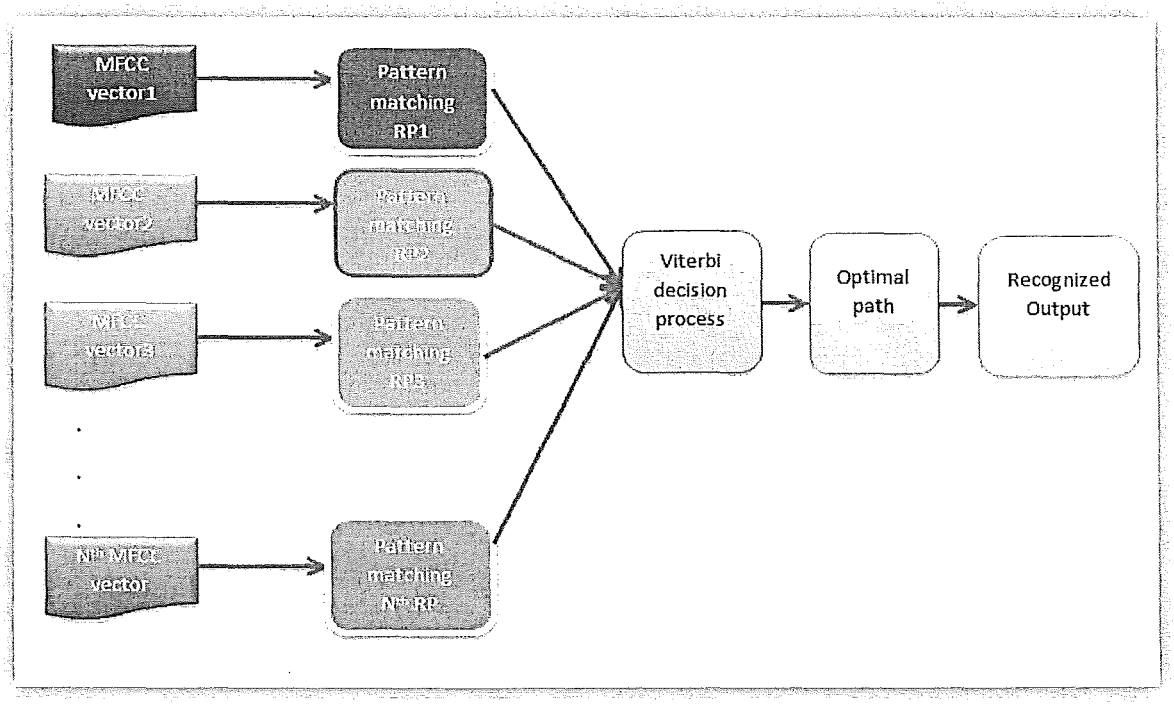

Figure 4.3: GPU implementation of Viterbi Search Algorithm

4.7. Performance Metrics

The following performance metrics were used to evaluate the performance of our implementations.

Word Accuracy Rate

$$WAR = \frac{N-S-D-I}{N} \quad \text{..... (5)}$$

Real Time Factor

$$RTF = \frac{P}{I_D} \quad \text{..... (1)}$$

According to Amdahl's Law the amount of parallel speedup in a given problem is limited by the sequential portion of the problem.

$$\textbf{Speedup} = \frac{t_s}{ft_s + \frac{(1-f)\ t_s}{p}} = \frac{1}{f + \frac{(1-f)}{p}} \qquad \lim_{p \to \infty} S(P) = \frac{1}{f} \quad \text{..... (14)}$$

4.8. Performance Results

This section presents the results of the implementations

Tables: 4.3 and 4.4, show the results of the implementations with regard to the running time, system accuracy, real time factor and the data size.

Table 4.3: Execution times and speedups of the implementations

Data Size	Running time (ms)			Speedup		
	CPU	CPU Optimised	GPU	CPU	CPU Optimised	GPU
64 ×64	13,03	5,38	1,88	1	2	7
100 × 100	26,75	11,23	6,09	1	2	4
400 × 400	575,60	216,79	34,27	1	3	17
900 × 900	2084,54	632,93	89,47	1	3	23
1600 × 1600	5259,16	1023,44	117,93	1	5	45
Average	7559.08	1889.77	249,64	1	4	30

4.4: The performance table of the implementations

	CPU	CPU Optimised	GPU
Runtime (ms)	7559.08	1889.77	249,64
WAR	78	76	83
RTF	12,75	8,83	4,25

Figures 4.4 to 4.9 depict the speech waveform, acoustics and spectrum for both the CPU and GPU implementations. The input is the same i.e. for the twentieth time that evening the two men shook hands.

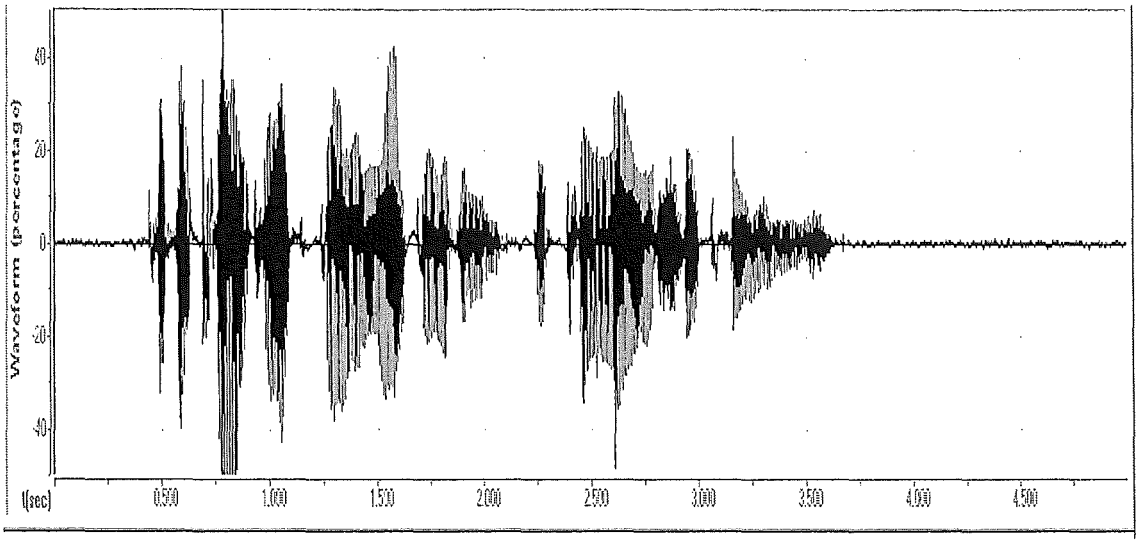

Figure 4.4: Speech waveform for CPU implementation

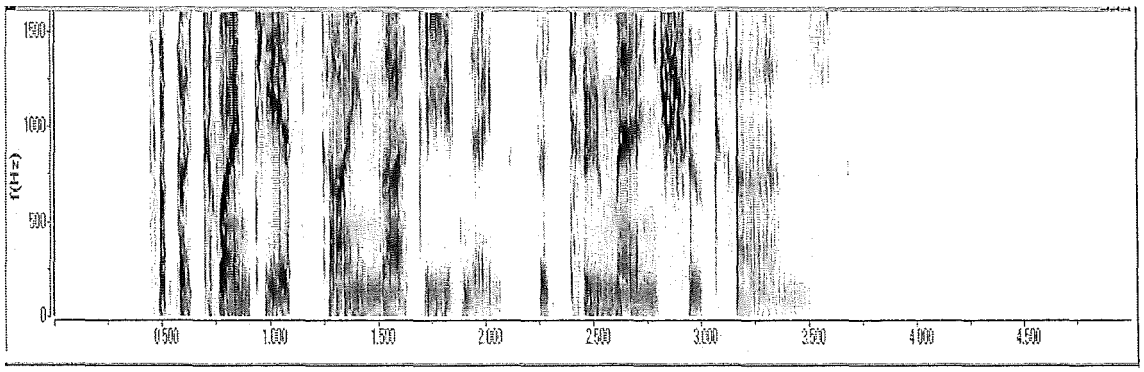

Figure 4.5: Speech acoustics for CPU implementation

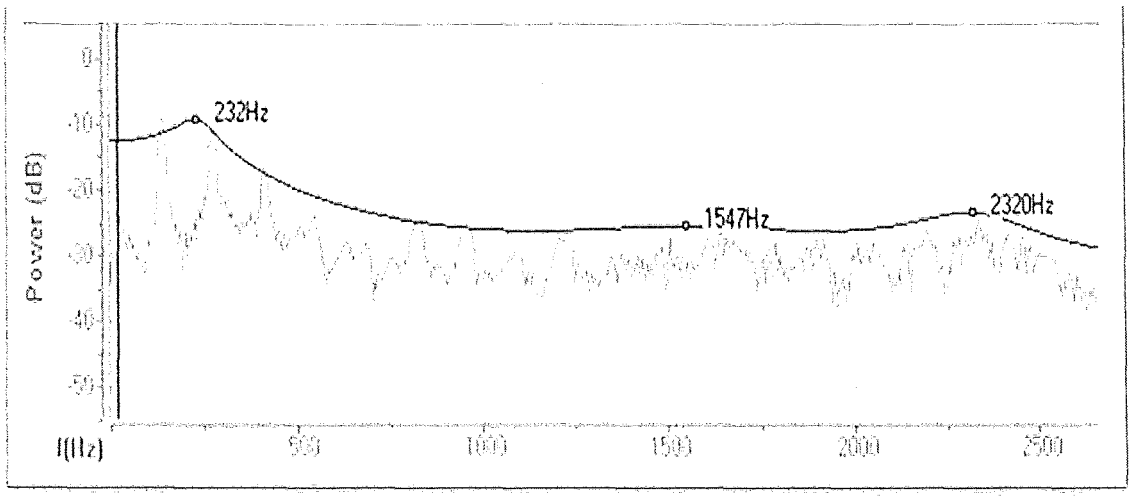

Figure 4.6: Spectrum for CPU implementation

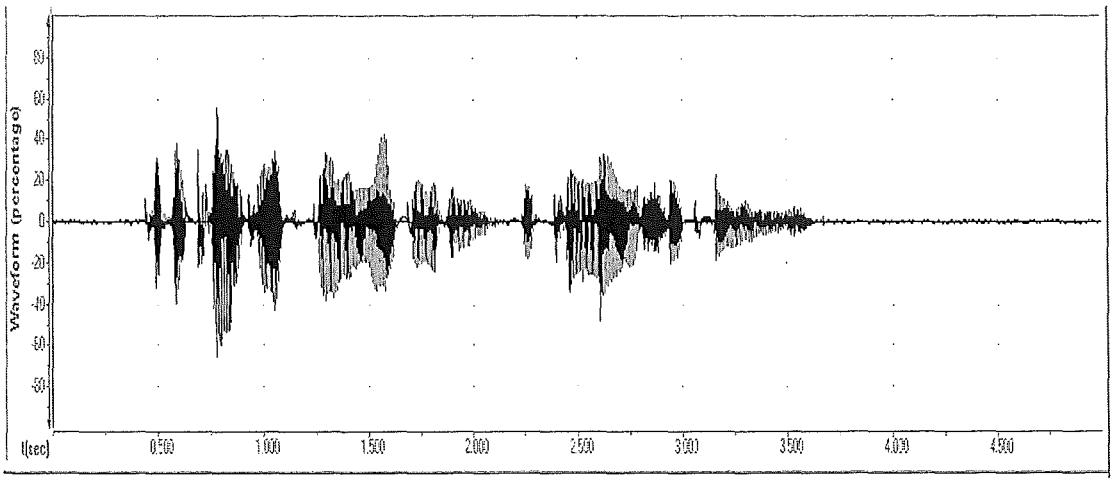

Figure 4.7: Speech waveform for GPU implementation

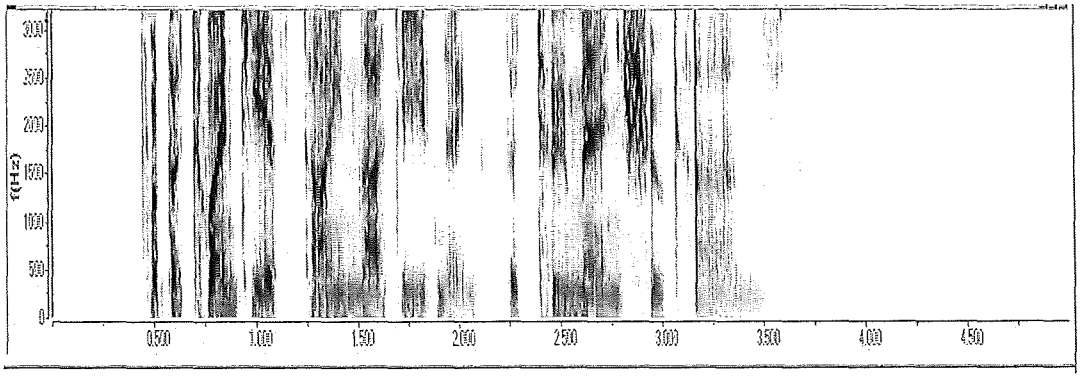

Figure 4.8: Speech acoustics for GPU implementation

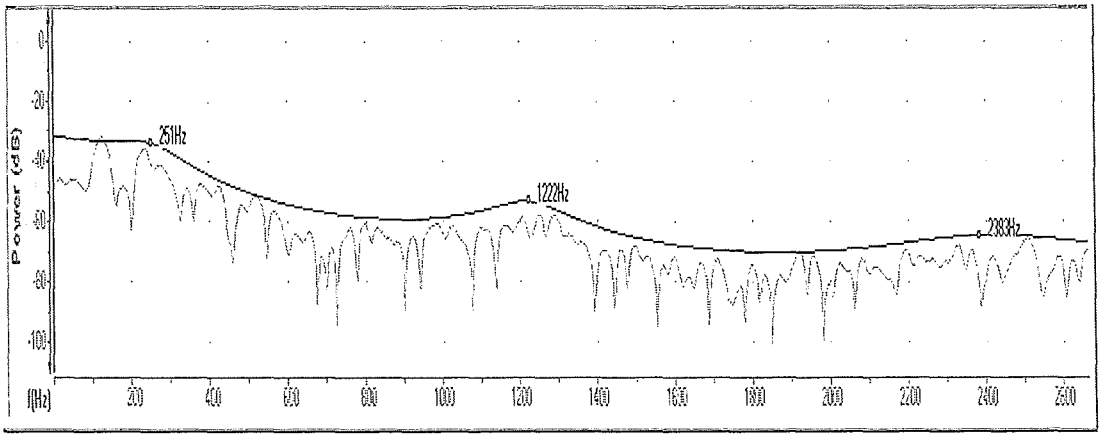

Figure 4.9: Spectrum for GPU implementation

Figure 4.4 and figure 4.7 depict the processing of the speech waveform of the same utterance on the CPU and GPU. The utterance was the same but the waveforms are somewhat different. The processing Frequencies of the waveform are different in the two platforms. The CPU was processing at 232Hz frequency whereas the GPU was processing at 251Hz frequency as depicted in Figure 4.6 and Figure 4.9. Contrary to the

observations on Figures 4.4 and 4.7, Figures 4.5 and 4.8 show that the acoustics are the same for both the CPU and GPU speech waveforms.

Figure 4.10 shows the performance graph for the runtime between the CPU, CPU optimised and GPU optimised implementations. According to the graph the runtime of the three implementations increases with the increase in data size.

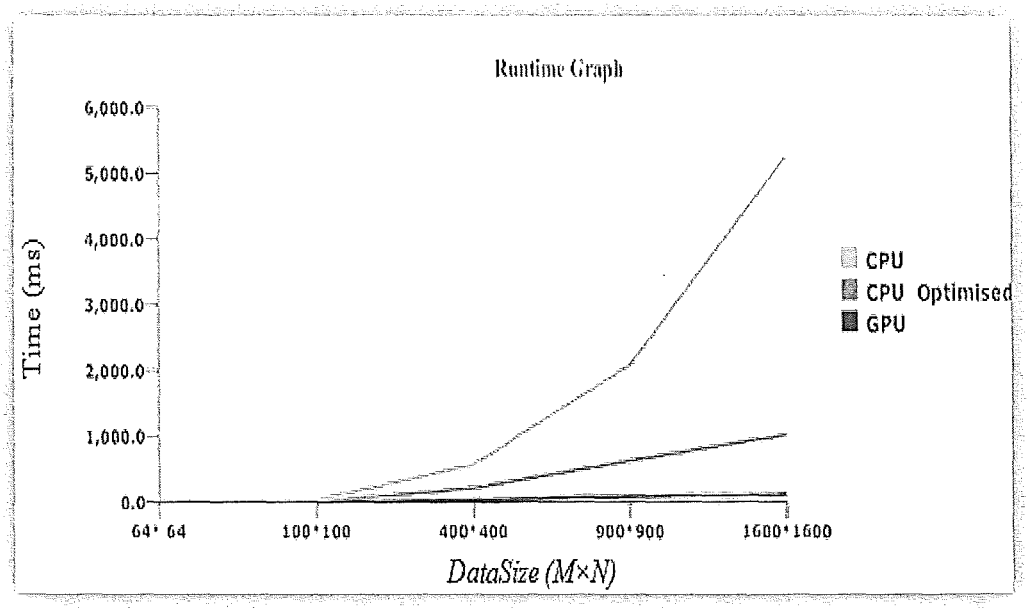

Figure 4.10: Performance runtime graph

4.9. Evaluation and Validation

This section discusses how the system performance was evaluate and validated.

We evaluated the performance of our C++ speech recognition Viterbi search algorithm and Viterbi search algorithm with the application of loop unrolling on an Intel core i7-3370 CPU running on Linux 12.0 Lt at 3.4GHz with 4 GB of RAM and our CUDA implementation on NVidia GeForce GTX 8800 GPU. We validated the results by comparing the outputs of the three implementations to verify their accuracy and efficiency. In this research work, the GPU implementations of our speech recognition systems gave an overall speech recognition accuracy of up to 83 %. Even though in this work these results were recorded as the best achievement compared to the CPU implementation, according to the literature best achievements were realized on Hachkar et al [**51**]which achieved a speech recognition accuracy of up to 92%. Other great achievements were also recorded on the work by Pour and Farokhi [**56**]which reported an average recognition accuracy of 98%.

Chapter 5: Summary and Concluding Remarks

5.1. Introduction to the Chapter

This chapter summarizes the research. It encompasses the aims and objectives that were achieved in the research, the discussion of the work done, the challenges faced during the course of the research, suggestions for future work and finally a summary of the dissertation.

5.2. Goals and Objectives of the Research from Chapter 1

Recall from chapter 1 that the goal of the research was:

To optimize the dynamic programming Viterbi search for automatic speech recognition.

The Objectives were:

A. To conduct a study on already existing language corpora, search algorithms and speech recognition systems.

B. To implement a speech recognition dynamic programming based Viterbi search algorithm.

C. To optimise dynamic programming based Viterbi search algorithm on a GPU based platform using CUDA and analyse the performance of the implementations.

In this chapter, we give an analysis of how the aims and objectives were attained and also explain the failure to reach some of the objectives.

5.3. Analysis of goals and objectives and Discussions

This section analyses the goal and objectives and also gives a brief discussion of the results.

5.3.1. Analysis of goals and objectives

The goal of this research was to optimize the dynamic programming Viterbi search for automatic speech recognition.

To achieve the goal, we carried out a literature study on automatic speech recognition, existing speech corpora, dynamic programming algorithms, speech recognition on GPUs, and HTK. Then we chose to use the Viterbi algorithm because it is simple to implement by means of Hidden Markov Models. We also chose to use the CMU US BDL Arctic speech corpus to train the system.

The Viterbi search algorithm was implemented on the Intel core i7 3370 CPU running on Linux 12.0 Lt at 3.4GHz with 4 GB of RAM. The performance was observed. The Viterbi search algorithm was optimised by applying loop unrolling then its performance was analysed. The Viterbi search algorithm with loop unrolling was optimised on the GPU based platform using CUDA. We synchronised the GPU threads by applying barrier synchronisation primitive. The implementation is very efficient and achieved a 30× speedup. The performance results of this implementation were analysed and compared with the two implementations that were performed on the CPU.

5.3.2. Discussions

We reported on the best runtime of the three implementations. There are typically substantial differences between the best and worst performing implementation.

Table 4.3 shows the runtime of the three implementations and indicates that the CPU implementation of the original Viterbi search algorithm is the worst performing implementation with regards to efficiency, but it performs much better than the CPU implementation of the Viterbi search algorithm with loop unrolling when it comes to word accuracy. The CUDA implementation is the best performing implementation out of the three. It achieved a better word accuracy rate and the best runtime. The CPU implementation of the Viterbi search algorithm with loop unrolling performed better than the CPU implementation of the original Viterbi search algorithm.

According to the results in Table 4.4, out of all the three implementations the original CPU implementation of the Viterbi search algorithm was the least efficient implementation, as the execution time increased as the data size (target matrix size) increased. The CPU implementation of the Viterbi search algorithm with loop unrolling performed 4× faster than the original implementation but its accuracy level decreased when compared to the original implementation. The GPU optimised implementation is the most efficient of the three. It achieved a performance speedup of 30× when compared to the original CPU implementation of the Viterbi Search and 8× speedup when compared to the Viterbi search algorithm with loop unrolling implementation.

5.4. Research challenges

The greatest challenges faced during the course of this research were lack of resources and the late arrival of some of the resources that were used.

5.5. Future work

In the future we want to implement a speech recognition application for mobile games and mobile applications. We would also compare the performance of loop unrolling and other loop transformation techniques on the Viterbi search algorithm. A combination of four or more loop transformation techniques to optimise the performance of the search process could make a great contribution to the future work on search process optimisation. Since the search process is such a computation and communication intensive task, we would like to perform more optimizations that will ensure system accuracy and efficiency that will lead to reliable real time systems.

5.6. Summary of dissertation

Speech recognition has progressed over the years. It is used in banking systems and translations systems among other application areas hence the efficiency of speech recognition is of importance. In this research we optimised the performance of the speech recognition Viterbi search. We implemented the Viterbi search algorithm on a CPU platform and found that most of the execution time of the Viterbi search is spent on loop iterations and this makes the search process a computation and communication intensive task. We took the algorithm and applied loop unrolling on the CPU platform and optimised the Viterbi search algorithm with the application of loop unrolling on a GPU platform. The GPU optimised implementation achieved a 30× speedup over the original CPU implementation and 8× speedup over the CPU implementation with the application of loop unrolling whereas the CPU implementation with the application of loop unrolling achieved a 4× speedup over that of the original CPU implementation of the Viterbi search algorithm.

Bibliography

[1] N. Indurkhya and F. J. Damerau, "An Overview of Modern Speech Recognition," in *Handbook of natural language processing*. London: CRC Press, 2009, ch. 15, pp. 339-366.

[2] M. Rahman, F. Khan, and A. Bhuiyan, "Continuous Bangla speech segmentation, classification and feature extraction.," *International Journal of Computer Science Issues, (IJCSI)*, vol. 9, no. 2, pp. 67-75, March (2012).

[3] What is an acoustic model? Voxforge. [Online]. http://www.voxforge.org/home/docs/faq/faq/what-is-an-acoustic-model

[4] Acoustic Modelling. Microsoft Research. [Online]. http://research.microsoft.com/en-us/projects/acoustic-modeling/

[5] What is an utterance? SIL International. [Online]. http://www01.sil.org/linguistics/GlossaryOfLinguisticTerms/WhatIsAnUtterance.htm

[6] D. Yook. (2003) Introduction to Automatic Speech Recognition: KOREA UNIVERSITY, Seoul, Lecture Notes. Document.

[7] K.M. Modieginyane,Z.P. Ncube and N. Gasela. "CUDA based performance evaluation of the computational efficiency of the DCT image compression technique on both the CPU and GPU," *Advanced Computing: An International Journal, (ACIJ)*, vol. 4, no. 3, May 2013.

[8] M. T. Dashti and A. J Wijs. (2007) Pruning state spaces with extended beam search. Document.

[9] J. Bilmes. (2005, January) Computer Speech Processing. Document.

[10] A.Yi and O Talakoub. (2009, April) Implementing a speech recognition system on a GPU using CUDA. document.

[11] O.P. Prabhakar and N. Kumar Sahu, "A Survey On: Voice Command Recognition Technique," *International Journal of Advanced Research in Computer Science and*

Software Engineering, vol. 3, no. 5, pp. 576-585, may 2013.

[12] M. De Wachter, M. Matton, K. Demuynck, P. Wambacq, R. Cools and D. Van Compernolle, "Template-Based Continuous Speech Recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 4, pp. 1377-1390, may 2007.

[13] M. Gales and S. Young, "The Application of Hidden Markov Models," *Foundations and Trends in Signal Processing*, vol. 1, no. 3, pp. 195–304, February 2008.

[14] S. Katagiri, *Pattern Recognition in Speech and Language Processing.*, 2003rd ed., W. Chou and B.-H.Juang, Ed. Basking Ridge, New Jersey: CRC Press, 2003.

[15] K.H.Davis, R.Biddulph, and S.Balashek, "Automatic Recognition of spoken Digits," *Journal of the Acoustical Society of America*, vol. 24, no. 6, pp. 637-642, November 1952.

[16] H.F.Olson and H.Belar, "Phonetic Typewriter," *Journal of the Acoustical Society of America* , vol. 28, no. 6, pp. 1072-1081, 1956.

[17] D.B.Fry, "Theoritical Aspects of Mechanical speech Recognitionand P.Denes, The design and Operation of the Mechanical Speech Recognizer at Universtiy College London," *Journal of British Institution of Radio Engineering*, vol. 19, no. 4, pp. 211-299, 1959.

[18] J.W. Forgie and C.D. Forgie, "Results obtained from a vowel recognition computer program," *Journal of the Acoustical Society of America*, vol. 31, no. 11, pp. 1480-1489, 1959.

[19] J. Suzuki and K. Nakata, "Recognition of Japanese vowels - preliminary to the recognition of speech," *J. Radio Res. Lab*, vol. 37, no. 8, pp. 193-212, 1961.

[20] T. Sakai and S. Doshita, "The phonetic typewriter.information processing," in *Proc.IFIP Congress*, Munich, 1962.

[21] Y.Kato, and S.Chiba K.Nagata, "Spoken Digit Recognizer for the Japanese Language," *Journal of the Audio Engineering Society* , vol. 12, no. 4, pp. 336,338, 340, 342, October

1964.

[22] T.B.Martin, A.L.Nelson, and H.J.Zadell, "Speech Recognition by Feature Abstraction Techniques," Air Force Avionics, Technical Report AL-TDR-64-176, 1964.

[23] T.K.Vintsyuk, "Speech Discrimination by Dynamic Programming," *Kibernetika*, vol. 4, no. 2, pp. 81-88, January-February 1968.

[24] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acoustics, Speech, Signal Processing (ASSP)*, vol. 26, no. 1, pp. 43-49, 1978.

[25] D. R. Reddy, "An approach to computer speech recognition by direct analysis of the speech wave.," Stanford University, Technical Report AD0640836, 1966.

[26] F.Itakura, "Minimum Prediction Residula Applied to Speech Recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing (ASSP)*, vol. 23, no. 1, pp. 67-72, February 1975.

[27] C.C.Tappert, N.R.Dixon, A.S.Rabinowitz, and W.D.Chapman, "Automatic Recognition of Continuous Speech Utilizing Dynamic Segmentation, Dual Classification,Sequential Decoding and Error Recover," Rome Air Development Centre, Rome, Technical Report 1971.

[28] F. Jelinek, L.R. Bahl and R. L. Mercer, "Design of a linguistic statisticaldecoder for the recognition of continuous speech," *IEEE Transactions on Information Theory*, vol. IT 21, no. 3, pp. 250-256, May 1975.

[29] C.H. Lee, L.R. Rabiner, R. Pieraccini and J.G. Wilpon, "Acoustic modeling for large vocabulary speech recognition," *Computer Speech and Language*, vol. 4, no. 2, pp. 127–165, April 1990.

[30] D. Klatt, "Review of the ARPA speech understanding project.," *Journal of the Acoustical Society of America*, vol. 62, no. 6, pp. 1324-1366, December 1977.

[31] B. Lowerre, "The HARPY speech understanding system," in *Trends in Speech*

Recognition. San Francisco, 1990, pp. 576-586.

[32] A. Lipeika, J. Lipeikiene, L. Telksnys, "Development of Isolated Word Speech Recognition System," *INFORMATICA*, vol. 13, no. 1, pp. 37–46 , 2002.

[33] A. Waibel, T.Hanazawa, G. Hinton, K. Shikano and K.J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Transactions on Acoustics Speech and Signal Processing*, vol. 37, no. 3, pp. 328-339, March 1989.

[34] S. Furui, "50 years of Progress in speech and Speaker Recognition Research," *ECTI Transactions on Computer and Information Technology*, vol. 1, no. 2, pp. 64-74, November 2005.

[35] Y. L. Chow, M. Dunham, O. Kimball, M. Krasner, G. Kubala, J. Makhoul, P. Price, S. Roucos and R Schwartz, "BYBLOS, the BBN continuous speech recognition system," in *Acoustics, Speech and Signal Processing, IEEE International Conference on ICASSP '87* ', Dallas, 1987, pp. 89-92.

[36] M. Weintraub, H. Murveit, M. Cohen, P.I. Price, J. Bernstein, G. Baldwin and D. Bell "Linguistic constraints in hiddenMarkov model based speech recognition.," in *Proceedings of ICASSP*, Glasgow, 1989, pp. 699-702.

[37] D. B. Paul, "The Lincoln robust continuous speech recognizer," in *Proceedings of ICASSP*, Glasgow, 1989, pp. 449-452.

[38] V. Zue, J. Glass, M. Phillips, and S. Seneff "The MIT summit speech recognitionsystem, a progress report," in *Proceedings of DARPA Speech and Natural Language Workshop*, Philadelphia, 1989, pp. 179-189.

[39] G. Poli, A. L. M. Levada, J. F. Mari, and J. H.Saito, "Voice Command Recognition with Dynamic Time Warping (DTW) using Graphics Processing Units (GPU) with Compute Unified Device Architecture (CUDA)," in *proceeding of: 19th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2007)*, Gramado, 2007.

[40] P. Cardinal, P. Dumouchel, G. Boulianne, and M. Comeau "GPU accelerated acoustic likelihood computations," in *Proceddings of the 9th international conference of*

interspeech, Brisbane, 2008, pp. 964-967. [Online]. http://www.crim.ca/Publications/2008/documents/plein_texte/PAR_CarPals_Interspeech 2008.pdf

[41] K. Vesely, L. Burget and F. Grezl "Parallel Training of Neural Networks for Speech Recognition," in *Interspeech*, Chiba, 2010, pp. 2934-2938.

[42] P.R. Dixon, T. Oonishi and S. Furui "Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition," *Computer Speech and Language*, pp. 510–526, March 2009.

[43] P. Cardinal, G. Boulianne and P.Dumouchel "THE A* SPEECH RECOGNITION SYSTEM ON PARALLEL ARCHITECTURES," in *Information Science, Signal Processing and their Applications, (ISSPA 2012)*, Montreal, 2012, pp. 108-113.

[44] M. K. Rehman, M. U. Sarwar, M. R. Talib, M. S. Mansoor and M.B. Sarwar "Parallel Implementation of Dynamic Programming Algorithm Using Graphics Processing Unit," *International Journal of Computer Science and Management Research*, vol. 2, no. 4, pp. 2097-2107, April 2013.

[45] Z Zhang Y Si and Y Yan Y Liu, "Accelerate Acoustic Likelihood Computations on GPU for Speech Recognition," in *proceedings of the 2nd international conference 0n computer science and electronics engineering*, Hangzhou, 2013, pp. 1019-1022.

[46] J. Chong, Y. Yi, E. Gonina, C. J. Hughes, Y. Chen, W. Sung, and K. Keutzer K. You, "Parallel Scalability in Speech Recognition," *IEEE Signal Processing Magazine*, pp. 124-135, November 2009.

[47] K. Gupta and J. D. Owens. (2010) Towards High-Quality Speech Recognition on Low-End GPUs.

[48] A. Hannani and T. Hain, "Automatic Optimization of Speech Decoder Parameters," *IEEE Signal Processing Letters*, vol. 17, no. 1, pp. 95-98, January 2010.

[49] O. Kalinli and S. Narayanan, "Continuous Speech Recognition Using Attention Shift Decoding with Soft Decision," in *Interspeech*, Brighton, 2009, pp. 1927-1930.

[50] E. Stoimenov and T. Schultz, "A Multiplatform Speech Recognition Decoder Based on Weighted Finite-State Transducers," in *Automatic Speech Recognition and Understanding, ASRU 2009*, Merano, 2009, pp. 293-298.

[51] A. Farchi , B.Mounir and J. EL Abbadi Z.Hachkar, "A Comparison of DHMM and DTW for Isolated Digits Recognition System of Arabic Language," *International Journal on Computer Science and Engineering ,(IJCSE)*, vol. 3, no. 3, pp. 1002- 1008, March 2011.

[52] T.B. Amin and I. Mahmood, "Speech Recognition Using Dynamic Time Warping ," in *ICAST 2008 2nd International Conference on Advances in Space Technologies*, Islamabad, 2008, pp. 74-79.

[53] H. Weisheng L. Wei, "Improved Viterbi Algorithm in Continuous Speech Recognition ," in *International Conference on Computer Application and System Modeling (ICCASM 2010)*, Taiyuan, 2010, pp. v7-207- v7-209.

[54] S. Ortmanns and H. Ney, "The Time-Conditioned Approach in Dynamic Programming Search for LVCSR," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 6, pp. 676-687, November 2000.

[55] J. Li and C. Lee, "Soft Margin Feature Extraction for Automatic Speech Recognition," in *Interspeech 2007*, Antwerp, 2007, pp. 30-33.

[56] F. Farokhi M. M. Pour, "An Advanced Method for Speech Recognition ," *World Academy of Science, Engineering and Technology* , vol. 3, no. 1, pp. 995-1000, January 2009.

[57] W. Chou and C. Lee B. Juang, "Minimum Classification Error Rate Methods for Speech Recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 5, no. 3, pp. 257-264, May 1997.

[58] L. Mesbahi and A. Benyettou M. Guezouri, "Speech Recognition Oriented Vowel Classification Using Temporal Radial Basis Functions," *Journal of Computing*, vol. 1, no. 1, pp. 162-167, December 2009.

[59] G. Linares, Y. Esteve and G. Gravier B. Lecouteux, "Dynamic Combination of

Automatic Speech Recognition Systems by Driven Decoding," *Journal of IEEE Transactions on Audio, Speech and Language Processing*, vol. 1, pp. 1-10, November 2012.

[60] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Word_error_rate

[61] M.A. Anusuya and S.K. Katti, "Speech Recognition by Machine: A Review," *(IJCSIS) International Journal of Computer Science and Information Security*, vol. 6, no. 3, pp. 181-205, September 2009.

[62] B Coppin, *Artificial intelligence illuminated.* Canada: Jones and Bartlett Publishers, 2004, pp. 73, 75, 76,108.

[63] B. Chen. (2006) search algorithms for speech recognition. Document. [Online]. http://berlin.csie.ntnu.edu.tw/Courses/Speech%20Recognition/Lectures2013/SP2013F_Lecture09_Search%20Algorithms.pdf

[64] J. McMichaels. (2003) Dynamic Time Warping & Search. Document.

[65] J. Eisner, "An Interactive Spreadsheet for Teaching the Forward-Backward Algorithm," in *Proceedings of the Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, Philadelphia, 2002, pp. 10-18.

[66] E. LaForest. (2010, March) Toronto University Website. [Online]. www.eecg.toronto.edu/~laforest/SurveyLoopTransformations.pdf

[67] C. Tadonki. (2010, december) University of Paris-Sud Website. [Online]. www.omegacomputer.com/staff/tadonki/PaperForWeb/tadonki_loop.pdf

[68] X. Zhang , Y. Zhang, X. Sun, F. Liu, S. Liu, Y. Tang and Y. Li. Automatic Performance Tuning of SpMV on GPGPU. Document.

[69] H. Mujtaba. (2013) WFCC tech. [Online]. http://wccftech.com/nvidia-tesla-k20-gk110-specifications-unveiled/

[70] N. Bell and M. Garland. (2009) Implementing Sparse Matrix-Vector Multiplication on

Throughput-Oriented Processors. Document.

[71] J. S. Godwin, "High-Performance Sparse Matrix-Vector Multiplication on GPUs For Structured Grid Computations," Ohio State University, Colombus, MSc Thesis 2013.

[72] B.B. Letswamotse, "The Performance Evaluation of Sparse Banded Matrix-Matrix Multiplication on CPU and GPU + CPU Heterogenous Environment Platforms," North-West University, Mafikeng, Honours Thesis 2012.

[73] F. Abi-Chahla. (2008 , June) Tomshardware. [Online]. http://www.tomshardware.com/reviews/nvidia-cuda-gpu,1954-6.html

Appendices

Appendix A: CPU Implementations of Viterbi Search Algorithm

```
/Viterbi loop/
for (currState = 0; currState < nStates; currState++) {
for (prevState = 0; prevState < nStates; prevState++) {
prob = prevState_dist[prevState] + transition[prevState*nStates + currState];

/applying loop unrolling to Viterbi search algorithm/
for (currState = 0; currState < nStates; currState+=3) {
for (prevState = 0; prevState < nStates; prevState+=3) {
prob = prevState_dist[prevState] + transition[prevState+0 *nStates+ currState+0]
prob = prevState_dist[prevState] + transition[prevState+1 *nStates+ currState+1]
prob = prevState_dist[prevState] + transition[prevState+2 *nStates+ currState+2];
```

Appendix B: GPU Implementation of Viterbi Search Algorithm

```
/Barrier Synchronisation primitive/

for (unsigned int stride = 1;stride <= blockDim.x; stride *= 2)

{
__syncthreads ( );
if (t % stride == 0)
PartialSum [2*t] + =PartialSum [2*t+ stride];
}
/Kernel for initialising index block/
__global__ void init_idx( int *idx_d, int nstates )
{
unsigned int idx = blockIdx.x * blockDim.x + threadIdx.x;
unsigned int idy = blockIdx.y * blockDim.y + threadIdx.y;
if (idx < nstates && idy < nstates) {
idx_d[(idy * nstates) + idx] = idx;
}
}
/makes use of shared memory/
```

```
index = globalId * localSize + localId;
for (currState = index; currState < nStates; currState += localSize*globalSize) {
for (prevState = 0; prevState < nStates; prevState++) {
if (prev % localSize == 0) {
cache[localId] = prevDist[prev + localId];
__syncthreads();
}
prob = cache[prevState % localSize] + transition[prevState + nStates*currState];
```

Appendix C: GeForce 8800

GeForce 8800 GTX has features that include Full Microsoft® DirectX® 10 Support, Unified Shader, 16X Anti-aliasing Technology, 128-bit Floating Point High Dynamic-Range (HDR) Lighting, GigaThread Technology and i-DSS – Support. Figure 4.1 gives a description of the GeForce 8800 architecture.

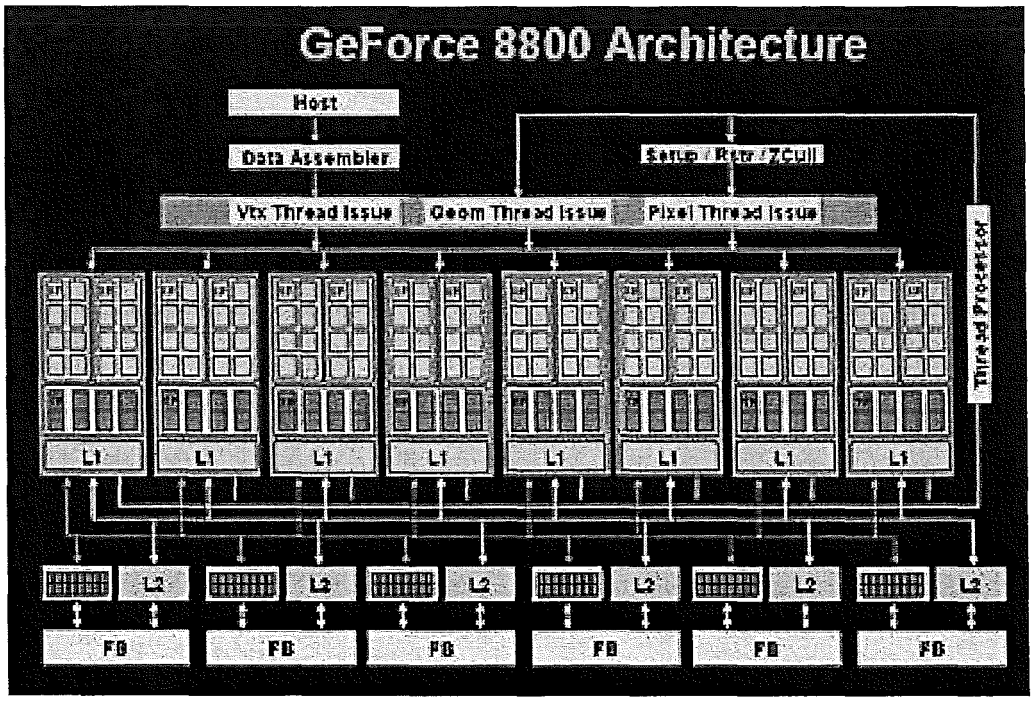

Figure C.1 GeForce 8800 architecture adapted from NVidia