# PARALLELISATION OF NAÏVE BAYES CLASSIFICATION FOR UNSTRUCTURED TEXT DOCUMENTS

**A DISSERTATION**

***Submitted in partial fulfillment of the requirements for the award of the degree***

***of***

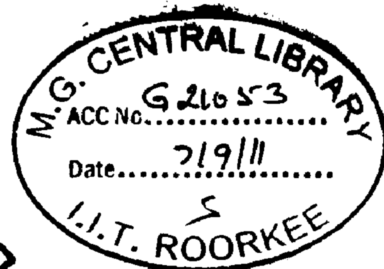
**MASTER OF TECHNOLOGY**

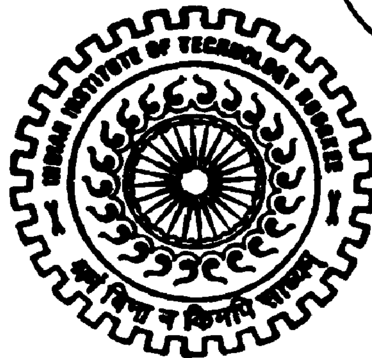
***in***

**INFORMATION TECHNOLOGY**

***By***

**NIKHIL AGRAWAL**



**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING**
**INDIAN INSTITUTE OF TECHNOLOGY ROORKEE**
**ROORKEE-247 667 (INDIA)**

**JUNE, 2011**

# CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled **"PARALLELISATION OF NAÏVE BAYES CLASSIFICATION FOR UNSTRUCTURED TEXT DATA"** towards the partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, Roorkee, Uttarakhand (India) is an authentic record of my own work carried out during the period from July 2010 to June 2011, under the guidance of **Dr. Durga Toshniwal, Assistant Professor,** Department of Electronics and Computer Engineering, IIT Roorkee.

The matter presented in this dissertation has not been submitted by me for the award of any other degree of this or any other Institute.

Date: 17/6/11

Place: Roorkee

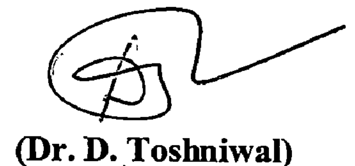
(NIKHIL AGRAWAL)

# CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

Date: 17/6/11.

Place: Roorkee



**(Dr. D. Toshniwal)**

Assistant Professor

Department of Electronics and Computer Engineering

IIT Roorkee.

# ACKNOWLEDGEMENT

First and foremost, I would like to extend my heartfelt gratitude to my guide and mentor **Dr. Durga Toshniwal**, Assistant Professor, Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee, for her invaluable advice, guidance, and encouragement and for sharing her broad knowledge. Her wisdom, knowledge and commitment to the highest standards inspired and motivated me. She has been very generous in providing the necessary resources to carry out my research. She is an inspiring teacher, a great advisor, and most importantly a nice person.

I am thankful for the useful comments and suggestions from faculty members of our institute. I am greatly indebted to all my friends, who have graciously applied themselves to the task of helping me with ample moral supports and valuable suggestions.

On a personal note, I owe everything to the Almighty and my parents. The support which I enjoyed from my father, mother and other family members provided me the mental support I needed.

**NIKHIL AGRAWAL**

# Abstract

Text classification is the task of assigning a given text document to one of the predefined categories depending on the contents of the document. It has found immense applications in fields as diverse as medicine, financial markets, information retrieval etc. Naïve Bayes' is one of the most widely used algorithms for classification. However, the algorithm is significantly slow due to the large amount of calculations it has to perform. Thus, there is a need to parallelise the algorithm to reduce the time required for classification. The algorithm could be parallelised using grid computing, clusters, CPU threads or GPUs.

Modern Graphics Processing Units (GPUs) have enabled high performance computing for general-purpose applications. GPUs are being used as co-processors in order to achieve a high overall throughput. CUDA programming model provides adequate C language like API, making it simpler to program for the GPU. In this dissertation, a CUDA based parallel implementation of Naive Bayes' text classification has been proposed. The classification step has been parallelised on GPU using different approaches each trying to exploit some property of the GPU. For example, use of shared memory against global memory, memory coalescing etc. The performance of the implementation of Naïve Bayes' text classification on GPUs has been compared with an efficient implementation of the same on a CPU.

The semantic information of unstructured text can be used to improve the classification accuracy. WordNet and POS tagging have been used in this dissertation, to capture the semantic information in unstructured text. The dataset used for experiments is Reuters-21578, which is a collection of news articles that appeared on the Reuters newswire in 1987. The proposed parallel Naïve Bayes' algorithm has been implemented on Nvidia's GTS 250 card with 128 processors and 512 MB GDDR3 RAM. The CPU used for the serial implementation consists of a Pentium P4 processor operating at 3 GHz and a DDR3 RAM of 4 GB. Experimental results show that the parallel implementation on GPUs is faster than the serial implementation.

# Table of Contents


**Candidate's Declaration and Certificate** i

**Acknowledgement** ii

**Abstract** iii

**Table of Contents** iv

**List of Figures** vii

**List of Tables** ix

**1. Introduction and Problem Statement** **1**

1.1 Introduction and Motivation 1

1.2 Statement of the Problem 3

1.3 Organization of the Report 4

**2. Background and Literature Review** **6**

2.1 Preprocessing 6

2.1.1 Feature Vector Generation 6

2.1.2 Stop Word Removal 6

2.1.3 Stemming 7

2.1.4 Semantic Information in Text 8

2.2 WordNet 8

2.3 Part of Speech tagging 9

2.4 Naïve Bayes' text classification 10

2.4.1 Multivariate Bernoulli Model 10

2.4.2 Multinomial Model 12

2.5 General Purpose Computing on GPUs 13

2.5.1 GPU Architecture 14

2.5.2 CUDA programming model 15

2.6 Literature Review 17

2.6.1 Parallelisation of Text Classification algorithms using CUDA 17

2.6.2 Parallelisation of Naïve Bayes' classification algorithm 18

2.6.3 Text representation using semantics 19

2.7 Research Gaps 20

**3. Proposed Framework 21**

3.1 Preprocessing Module 22

3.2 Training Module 22

3.3 Serial Naïve Bayes' classification 23

3.4 Parallel Naïve Bayes' classification 23

**4. Detailed Design and Implementation 24**

4.1 Preprocessing Module 24

4.2 Training Module 29

4.3 Classification Module 30

4.4 Experimental Setup 36

4.4.1 Data Preparation 36

4.4.2 Hardware Configuration 37

**5. Results and Discussions 39**

5.1 Comparison of classification accuracy 39

5.1.1 Results of Integrating WordNet and POS tagging 39

5.2 Results of Parallelisation 40

5.2.1 Determining the number of threads 40

5.2.2 Comparison of parallel Naïve Bayes' with serial Naïve Bayes' 42

5.2.3 Comparison between the two parallel Naïve Bayes' implementations 44

**6. Conclusions and Future Work 47**

6.1 Conclusions 47

6.2 Scope for Future Work 47

**References 49**

**List of Publications 52**

# LIST OF FIGURES


| | | |
|---|---|---|
| Figure 2.1 | Floating Point operations for GPU and CPU | 13 |
| Figure 2.2 | GPU devotes more transistors to data processing | 14 |
| Figure 2.3 | Thread hierarchy in CUDA programming model | 15 |
| Figure 2.4 | Memory hierarchy in CUDA programming model | 16 |
| Figure 2.5 | CUDA software stack | 17 |
| Figure 3.1 | System architecture of Naïve Bayes' text classification | 21 |
| Figure 4.1 | Block diagram of Preprocessing Module | 24 |
| Figure 4.2 | Flow chart of the preprocessing module using POS tagging and WordNet synsets | 25 |
| Figure 4.3 | Flow chart of the GetIndex subroutine using POS tagging and WordNet synsets | 26 |
| Figure 4.4 | Flow chart of the preprocessing module using WordNet synsets | 27 |
| Figure 4.5 | Flow chart of the GetIndex using WordNet synsets | 28 |
| Figure 4.6 | Flow chart of the Training module of Naïve Bayes' text classification | 29 |
| Figure 4.7 | Model for parallel Naïve Bayes' text classification module | 31 |
| Figure 4.8 | Representation of class-conditional word probabilities, feature vector and class probability | 32 |
| Figure 4.9 | Processing done by a GPU block | 32 |
| Figure 4.10 | Memory access strategies for class-conditional word probability matrix and feature vector | 34 |

Figure 4.11 Addresses generated by threads in a warp 34

Figure 4.12 Processing done to obtain the final class probability 36

Figure 5.1 Comparison of execution time of the two parallel Naïve Bayes' implementations on GPU by varying the number of threads/block 41

Figure 5.2 Comparison of execution time of Naïve Bayes' using CPU, GPU (sequential memory accesses) and GPU (interleaved memory accesses) by varying number of classes 43

Figure 5.3 Speed up obtained by using GPU as compared to CPU 44

Figure 5.4 Comparison of kernel execution time of Naïve Bayes' GPU (sequential memory accesses) and GPU (interleaved memory accesses) by varying number of classes 46

# LIST OF TABLES


| | | |
|---|---|---|
| Table 4.1 | Hardware specification of GTS 250 card | 37 |
| Table 4.2 | Hardware specifications of the CPU used | 38 |
| Table 5.1 | Comparison of classification accuracy of proposed Naïve Bayes' text classification with that of WEKA | 39 |
| Table 5.2 | Comparison of classification accuracy obtained by using Wordnet and POS tagging | 40 |
| Table 5.3 | Comparison of the execution time of the two parallel Naïve Bayes' implementations on GPU by varying the number of threads/block | 41 |
| Table 5.4 | Comparison of the execution time by varying the number of classes | 42 |
| Table 5.5 | Comparison of Execution Time of the two parallel implementations of Naïve Bayes' | 45 |

# Chapter 1

# Introduction and Statement of the Problem

## 1.1 Introduction and Motivation

Text classification is the task of assigning a given text document to one of the predefined categories depending on the contents of the document. It has found immense applications in areas as diverse as medicine, e-commerce, information retrieval, financial markets etc. Traditionally the task of text classification was done by human experts and usually required a large amount of time. But, the rapid growth of information on the web has led to development of algorithms that could enable automatic text classification with accuracy similar to that of a human expert but taking a much lesser amount of time. Some of these algorithms are Support Vector Machines (SVM), k-nearest neighbours (kNN), Naïve Bayes' etc. These algorithms form a part of a broader category of algorithms called machine learning algorithms.

The text classification algorithms operate in 2-steps: *learning* and *classification.* In the *learning step* a model is built using already classified documents. *Classification step* uses the model built in the *learning step* to classify documents to one of the available classes. Formally the task of text classification can be stated as follows [1]: given a set of classification labels *C*, and a set of training text documents *E*, each of which has been assigned one of the class labels from *C*, the system must use *E* to develop a hypothesis that can be used to predict the class labels of previously unseen examples of the same type.

Before a set of documents can be presented to the system, each document must be converted to a feature vector because the classification algorithm or the classifier-building algorithm cannot interpret the text directly. Each element of the feature vector may represent a word or a phrase from the given text document. This representation of text could be used with many classification algorithms like SVM, kNN, Naïve Bayes' and so on. The values in the feature vector may be binary or integers.

Binary values just indicate the presence or absence of a term in the document whereas integer values indicate the frequency of term's occurrence in the document. This representation of text is termed as bag-of-words and has been used extensively by researchers and professionals alike. The information about the order of words is lost in this model but Joachims [2] refers that this loss is irrelevant since the information lost is little and bringing unnecessary complexity to the task of classification is questionable.

Using the bag-of-words approach, the dimensionality of the feature vector becomes very high; the number of features may go up to tens of thousands. This leads to increased computational and space complexity. There are a large variety of preprocessing steps for feature set reduction. Silva, C.; Ribeiro, B. [3] evaluated the performance of three major pre-processing steps, namely stop word removal, stemming and the removal of words with low document frequency. Their results showed that the use of the pre-processing steps did not lead to much degradation in classification accuracy. One drawback of the traditional bag-of-words model is that the semantic information like synonyms, antonyms, hyponyms, part-of-speech between the words in the text document is not captured and is lost. Hence, there is a need to develop systems where the semantic information of text is captured into the bag-of-words model.

Stop words are non-informative words such as articles, prepositions and conjunctions. These words have no distinguishing potential between the various categories. Stemming is another pre-processing step to avoid feature expansion. In stemming the word stem is derived from the occurrence of the word by removing case and inflection information. For example, *"computer"*, *"computes"* and *"computing"* are all mapped to the stem *"comput"*.

Naïve Bayes' is based on a probabilistic model for text classification. Given a text document, the algorithm assigns it to the class that is most probable to have generated the document. Over the years the algorithm has found immense applications like stock market prediction, information retrieval, e-mail spam filtering, heart-disease prediction system and many more. The reason for such a wide use of the algorithm is not only the simplicity of its learning step and the classification step, but also that it gives satisfactory classification accuracy. However, the algorithm is slow due to the

large amount of calculations it has to perform. Thus, there is a need to parallelise the algorithm using architectures like grid computing, clusters, CPU multi-threading, GPUs etc.

The amount of information that is processed and maintained has been increasing day by day. Moreover, a large part of this information is in the form of unstructured text. Text classification is an important task in maintaining the text data. There is a need to develop systems that could perform the task of text classification requiring as less time as possible. Thus, there is a need to parallelise the task of text classification.

Graphics Processing Unit, or GPU as it is popularly called, is a programmable logic chip that performs parallel computations on graphics data. Recently GPUs are being used for a variety of applications that require repetitive computations on multiple sets of data. With the advent of CUDA, a parallel programming architecture developed by Nvidia, it has become easier to program the GPU than it was possible earlier. CUDA enables the programmer to program using variants of high level programming languages like C and Java. As a result, there has been a lot of work towards parallelizing text classification algorithms using CUDA. The improvement in execution time, obtained by parallelization of classification algorithms like kNN and SVM, is a motivating factor for parallelisation of the Naïve Bayes' classification algorithm on the GPU.

## 1.2 Statement of the Problem

The problem statement of the dissertation is as follows:

*Parallelisation of Naïve Bayes' classification for unstructured text data.*

The above problem can be divided into the following sub-problems.

- Preprocessing Module: To convert a given document to a feature vector representation so that it can be used by the classification algorithm. It involves a lot of steps like POS tagging, tokenization, stop word removal and then finally generating the feature vector.

- Training Module: To determine the parameters, i.e., the class-conditional word probabilities and the class prior probability, of the Naïve Bayes' model, using the training dataset provided. This information is then used in the classification step.

- Classification Module: To classify a given test document, into one of the predefined categories, using the model built by the Training Module. Since this step is the one to be used recurrently, it is the one that has been parallelized.

## 1.3 Organization of the Report

This dissertation report comprises of six chapters including this chapter that introduces the topic and states the problem. The rest of the report is organized as follows.

Chapter 2 gives the background of preprocessing, WordNet and POS tagging, Naïve Bayes' text classification, GPU architecture and CUDA, literature review of text representation using semantics, parallelization of text classification algorithms using CUDA and parallelization of Naïve Bayes' text classification.

Chapter 3 discusses the proposed framework which includes the preprocessing module, training module and the parallel Naïve Bayes' classification module.

Chapter 4 gives the detailed design and implementation of all the modules of the proposed framework. It also gives a description about the experiments performed to evaluate the proposed model.

Chapter 5 discusses the experimental results, validation of the system, and comparison of the results obtained from different approaches.

Chapter 6 concludes the dissertation work giving the scope for future work.

# Chapter 2

# Background and Literature Review

## 2.1 Preprocessing

The text documents need to be processed to convert them to a representation understood by the Naïve Bayes' classification system. The preprocessing steps used in the proposed system are discussed below.

### 2.1.1 Feature Vector Generation

Classification algorithms cannot process the text directly in its raw form. Instead, the text documents need to be converted to a representation which can be understood by the classifier. One such representation is to use a binary vector, each element of which indicates the occurrence or absence of a term in the document. Another approach is the use of bag-of-word (BoW) model in which the frequency count of each term in the text document becomes an element of the feature-vector. The BoW model is also called the Vector Space Model.

One of the drawbacks of the BoW model is the high dimensionality of the feature-vector. To reduce the dimensionality to some extent stop word removal and stemming are used in almost all text classification applications. These techniques have been explained in the following sections. Another drawback of the BoW model is that it does not capture the semantic information that exists among the terms in the text document.

### 2.1.2 Stop word Removal

Text documents contain words like *'a'*, *'an'*, *"the"* etc, which do not convey any information about the document. These words occur in all documents irrespective of the class of the document. Such words are called stop words. These words should be removed from the text document before proceeding with the classification task or the results of classification could be misleading.

The most obvious method for stop word removal is to maintain a list of stop words and check each word in the given text for presence in the stop word list. If a word is present in the stop word list then the word is removed from the given text and not considered during classification. Also it should be kept in mind that there is no definitive list of stop words. The list depends on the application under consideration. This list could either by supplied by a human expert in the concerned domain or could be a standard one which are readily available.

### 2.1.3 Stemming

In stemming, the word stem is derived from the occurrence of the word by removing case and inflection information. The stem form need not be identical to the morphological root of the word as it is sufficient that related words reduce to the same stem even if the stem is not a valid root itself. E.g. *fishing* and *fisher* get reduced to *fish* which is the stem for those words.

Though stemming helps to reduce the feature set size, it has one drawback. The use of stemming can sometimes lead to errors in the classification task as it is possible that two unrelated words get reduced to the same stem, e.g. *universal* and *university* get stemmed to the same root; and *animal* and *animation* get stemmed to the same root. There exist several methods for stemming which are listed below:

- Brute Force Algorithms
- Suffix-stripping Algorithms
- Lemmatization Algorithms
- Stochastic Algorithms

Suffix-stripping Algorithms are highly popular because they are simple to implement and give satisfactory results. Suffix stripping algorithms do not rely on a lookup table that consists of inflected forms and root form relations. Instead, a typically smaller list of "rules" is stored which provides a path for the algorithm; given an input word form, to find its root form. Some examples of the rules include:

- if the word ends in 'ed', remove the 'ed'
- if the word ends in 'ing', remove the 'ing'
- if the word ends in 'ly', remove the 'ly'

Porter stemming algorithm is a suffix stripping algorithm published by Martin Porter in the July 1980 issue of the journal *"Program"*. It became immensely popular and kind of became a standard algorithm used for English language [4].

### 2.1.4 Semantic Information in Text

The terms in the text document are semantically related by a number of relations like synonymy, antonymy etc. Also the part-of-speech of the terms has a profound effect on the semantics of the term. The BoW model does not capture the semantics of the text document. Various approaches like ontology based, N-Grams, multi-word features, Latent Semantic Indexing (LSI), Locality Preserving Indexing (LPI) have been proposed and are in use over the years. In the *Literature Review* section, some of the techniques used for using the semantic information among the terms in the text document, have been discussed.

## 2.2 WordNet

WordNet is a lexical database for English language. English nouns, verbs, adjectives, and adverbs are organized into sets of synonyms, each representing a lexicalized concept [5]. WordNet does not include prepositions, determiners etc. The synsets are connected to other synsets via the following semantic relations [5]:

- *Synonymy* is WordNet's basic relation, because WordNet uses sets of synonyms (*synsets*) to represent word senses. Synonymy is a symmetric relation between word forms.

- *Antonymy* (opposing-name) is also a symmetric semantic relation between word forms, especially important in organizing the meanings of adjectives and adverbs.

- *Hyponymy* (sub-name) and its inverse, *hypernymy* (super-name), are transitive relations between synsets. Because there is usually only one hypernym, this semantic relation organizes the meanings of nouns into a hierarchical structure.
- *Meronymy* (part-name) and its inverse, *holonymy* (whole-name), are complex semantic relations. WordNet distinguishes *component* parts, *substantive* parts, and *member* parts.
- *Troponymy* (manner-name) is for verbs what hyponymy is for nouns, although the resulting hierarchies are much shallower.
- *Entailment* relations between verbs are also coded in WordNet.

WordNet also provides the *polysemy count* of a word: the number of synsets that contain the word. The morphology functions of the software distributed with the database try to deduce the lemma or root form of a word from the user's input; only the root form is stored in the database unless it has irregular inflected forms [5].

## 2.3 Part of Speech Tagging

A Part-Of-Speech Tagger (POS Tagger) is a piece of software that reads text in some language and assigns parts of speech to each word (and other token), such as noun, verb, adjective, etc., although generally computational applications use more fine-grained POS tags like 'noun-plural'. In this thesis the *Stanford Log-linear Part-Of-Speech Tagger* has been used. The tagger is available at [6]. The software requires Java 1.5+ to be installed. The tagger takes as input a string of words and returns a string consisting of words annotated with their part of speech. The word and part of speech pair is separated by a "/". The English taggers in the software use the Penn Treebank POS tag set. For example the sentence "*I, Nikhil Agrawal hereby declare that this thesis report is a copy of original work.*" would be tagged and returned as "*I/PRP , /, Nikhil/NNP Agrawal/NNP hereby/NN declare/VBP that/IN this/DT thesis/NN report/NN is/VBZ a/DT copy/NN of/IN original/JJ work/NN . /.*"

## 2.4 Naïve Bayes' Text Classification

Naïve Bayes' classification is a relaxed version of the Bayesian classifiers which assumes the features to be independent of each other. It is based on a probabilistic model for text classification. Given a text document *d*, the algorithm assigns it to the class that is most probable to have generated the document. The following equation [7] is used to calculate the probability of class *c*, given the document *d*.

$$P(c \mid d) = \frac{P(d \mid c)\, P(c)}{P(d)} \tag{2.1}$$

The estimation of $P(d|c)$ is difficult but with the naïve assumption that the features are independent of each other it is simply a running product of class-conditional word probabilities. Also since $P(d)$ is constant for all the classes, and we are interested in finding the maximum $P(c|d)$; $P(d)$ can be ignored from the calculations.

There exist a lot of models for Naïve Bayes' text classification like Multinomial model, Multivariate Bernoulli model, Poisson model etc. The Multivariate Bernoulli model and the Multinomial model are the most popular models of Naïve Bayes' text classification. The Poisson model of Naïve Bayes' text classification has not found many applications and hence will not be discussed in this report. The following sub-sections give a brief description of the two models.

### 2.4.1 Multivariate Bernoulli Model

In this model, the document is treated as a binary vector, each element of which indicates the presence or absence of a word in the document. The model does not record the actual number of occurrences of a word in the document. It is called the Multivariate Bernoulli model because it treats the document as a collection of many Bernoulli experiments, each corresponding to a word in the feature set. The following equations given by [8] are the most popularly used equations to implement the Multivariate Bernoulli model.

The probability of a document given its class [8]:

$$P(d_i \mid c_j;\theta) = \prod_{t=1}^{|V|} \left( B_{it} P(w_t \mid c_j;\theta) + (1 - B_{it})(1 - P(w_t \mid c_j;\theta)) \right) \tag{2.2}$$

- $B_{it}$ : *0/1* indicating absence/presence of word $w_t$ in document $d_i$
- $|V|$ : total number of words in the feature set (also called vocabulary size)
- $P(w_t \mid c_j;\theta)$ : Probability of word $w_t$ given class $c_j$

The estimate for class-conditional word probabilities [8]:

$$\theta_{w_t|c_j} = P(w_t \mid c_j;\theta) = \frac{1 + \sum_{i=1}^{|D|} B_{it} P(c_j \mid d_i)}{2 + \sum_{i=1}^{|D|} P(c_j \mid d_i)} \tag{2.3}$$

- $P(c_j \mid d_i)$ : *0* if document $d_i$ does not belong to class $c_j$

  *1* otherwise
- The addition of one and two in the numerator and denominator respectively is done to avoid counts of zero or one.

The class prior probabilities are given by [8]:

$$\theta_{c_j} = P(c_j;\theta) = \frac{\sum_{i=1}^{|D|} P(c_j \mid d_i)}{|D|} \tag{2.4}$$

The disadvantage of this model is that it has low classification accuracy as it does not capture the word counts. The model is less used as compared to the Multinomial model because of its lower classification accuracy. The next sub-section discusses the Multinomial model of Naïve Bayes' text classification.

### 2.4.2 Multinomial Model

Multinomial model uses the standard bag-of-words representation in which each element of the feature vector gives the frequency of the corresponding word in the document. The assumption that the occurrence of a word in the document is independent of its context still holds true. The model regards the occurrence of a word as an event. Thus, it can be said that each document $d_i$ is drawn from a multinomial distribution of words with as many independent trials as the length of $d_i$. The estimate of the probability of word $w_t$ given class $c_j$ is given by equation 2.5 [8]:

$$\theta_{w_t|c_j} = P(w_t \mid c_j;\theta) = \frac{1+\sum_{i=1}^{|D|} N_{it} P(c_j \mid d_i)}{|V| + \sum_{s=1}^{|V|}\sum_{i=1}^{|D|} N_{is} P(c_j \mid d_i)} \quad (2.5)$$

- $|D|$: Total number of documents in the training set
- $N_{it}$: Number of times word $w_t$ occurs in document $d_i$
- $|V|$: Total number of words in the dictionary (= Number of elements in the feature vector).
- $P(c_j \mid d_i)$ : $1$ if document $d_i$ belongs to class $c_j$; $0$ otherwise

The class prior probabilities for this model are calculated in the same way as for the Multivariate Bernoulli model (equation 2.4). Once the training information is available the classifier can be used to predict the class label of unknown documents using equation 2.1. The estimate for the probability of document given its class is calculated using equation 2.6 [8].

$$P(d_i \mid c_j;\theta) = P(|d_i|)\,|d_i|!\prod_{t=1}^{|V|}\frac{P(w_t \mid c_j;\theta)^{N_{it}}}{N_{it}!} \quad (2.6)$$

Then, substituting the results obtained from equation 2.6 and that of equation 2.4, in equation 2.1, we get the probability of a class given the document. After doing so for all the classes, the document is assigned to the class with the maximum probability.

It has been shown that the Multinomial model is more accurate than the Multivariate Bernoulli model [8]-[9]. This is because the Multinomial model unlike the Multivariate Bernoulli model captures the word counts when creating the document vector. Also the model has been found to be applicable to large datasets without degradation in performance. Thus, in this dissertation the Multinomial model has been used for the proposed parallel Naïve Bayes' text classification.

## 2.5 General Purpose Computing on GPUs

Driven by the insatiable market demand for real-time, high-definition 3D graphics, the programmable GPU has evolved into a highly parallel, multithreaded, multiple core processor with tremendous computational horsepower and very high memory bandwidth. Figure 2.1 [10] illustrates the computational power of the GPUs provided by Nvidia.

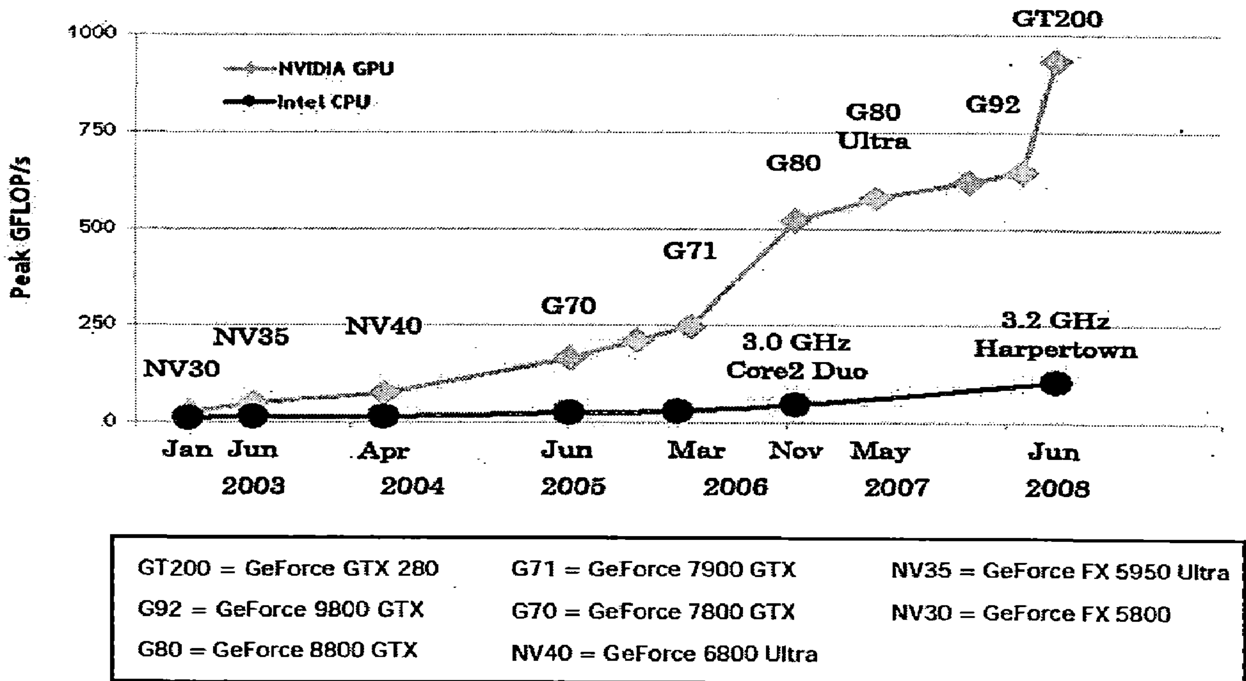



**Figure 2.1 Floating point operations for the GPU and CPU**

Moreover with the evolution of Nvidia's CUDA programming model it has become very easy to program for the GPU. This simplicity has led to a tremendous increase in the use GPUs for general purpose computing.

### 2.5.1 GPU Architecture

The reason behind the discrepancy in floating-point capability between the CPU and the GPU is that the GPU is specialized for compute-intensive, highly parallel computations and is designed such that more transistors are devoted to data processing rather than data caching and flow control, as schematically illustrated by Figure 2.2 [10].

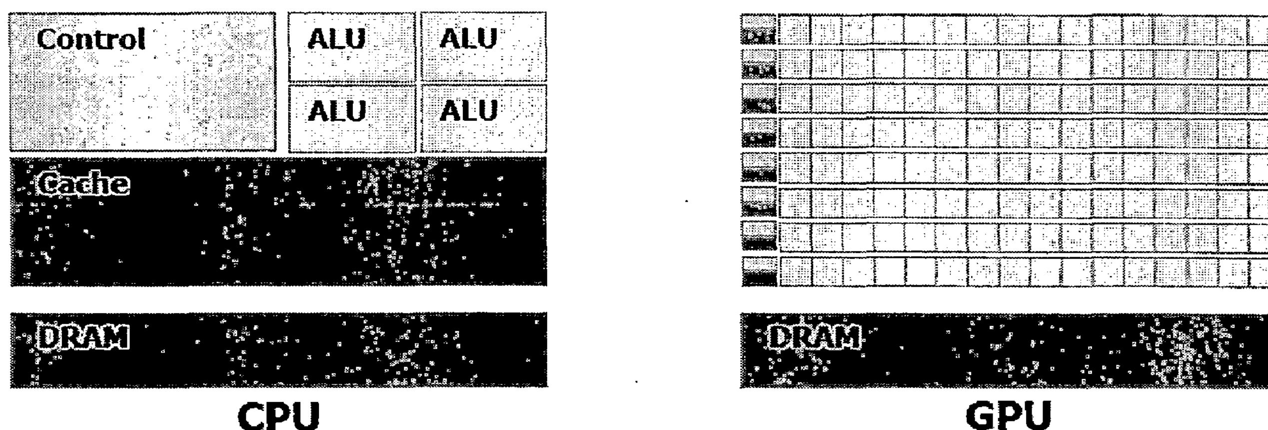



**Figure 2.2 GPU devotes more transistors to data processing**

More specifically, the GPU is especially well-suited to address problems that can be expressed as data-parallel computations, i.e., the same program is executed on many data elements in parallel. The operations must be one with high arithmetic intensity. Arithmetic intensity is defined as the ratio of arithmetic operations to the memory operations. Data-parallel processing maps data elements to parallel processing threads. Many algorithms like pattern matching, computational finance etc. that process large data sets can be accelerated by using the data-parallel programming model. The CUDA programming model exposes the parallel computing capabilities of the GPUs and has been a hot topic ofresearch recently.

### 2.5.2 CUDA programming model

CUDA (Compute Unified Device Architecture) is a comprehensive software and hardware architecture for GPGPU that was developed and released by Nvidia in 2007. CUDA supports heterogeneous computations in a sense that serial parts of an application are executed on the CPU and parallel parts on the GPU [11]. The CUDA programming model encourages dividing problems in two steps: At first into coarse independent sub-problems (grids) and afterwards into finer sub-tasks that can be performed cooperatively (thread blocks). The programmer writes a serial C for CUDA program which invokes parallel *kernels* (functions written in C) [11]. The kernel is usually executed in thousands of threads, which the programmer organizes in a hierarchy as shown in Figure 2.3 [11].

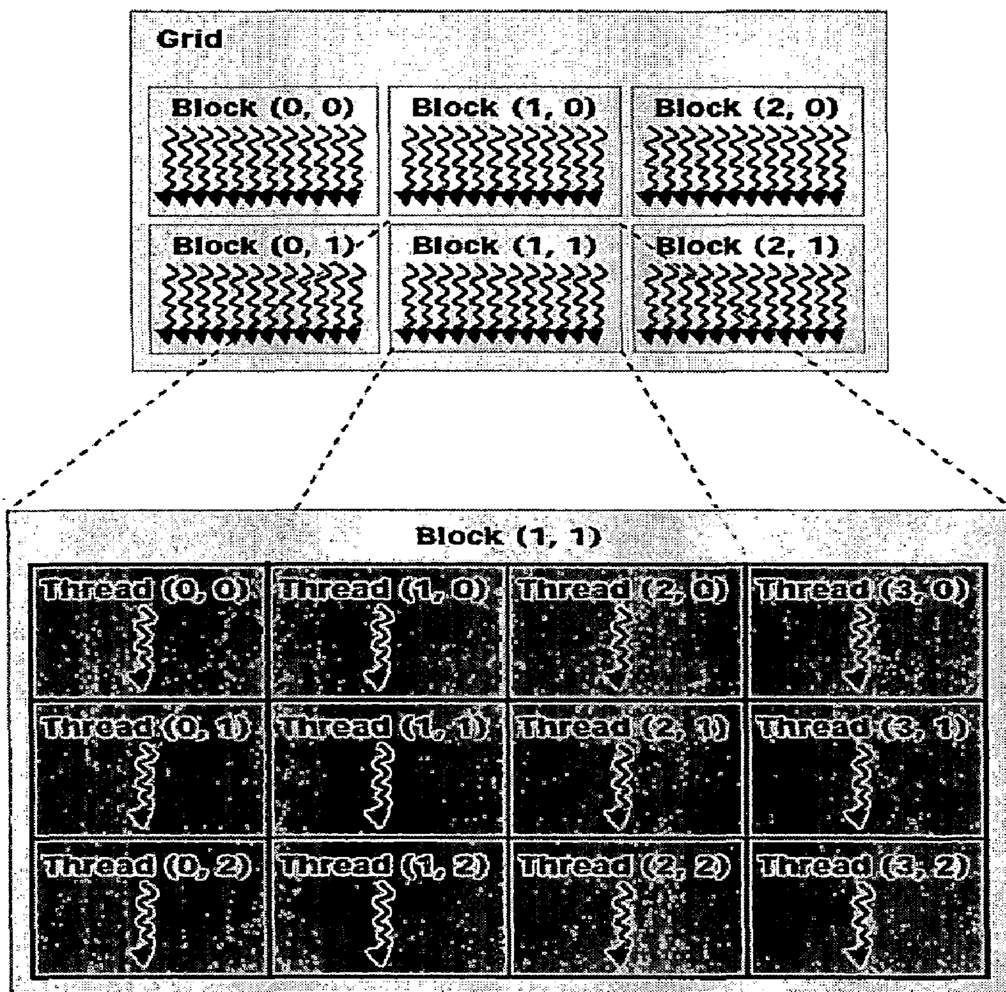



**Figure 2.3 Thread Hierarchy in CUDA programming model**

CUDA threads may access data from multiple memory spaces during their execution as illustrated by Figure 2.4.

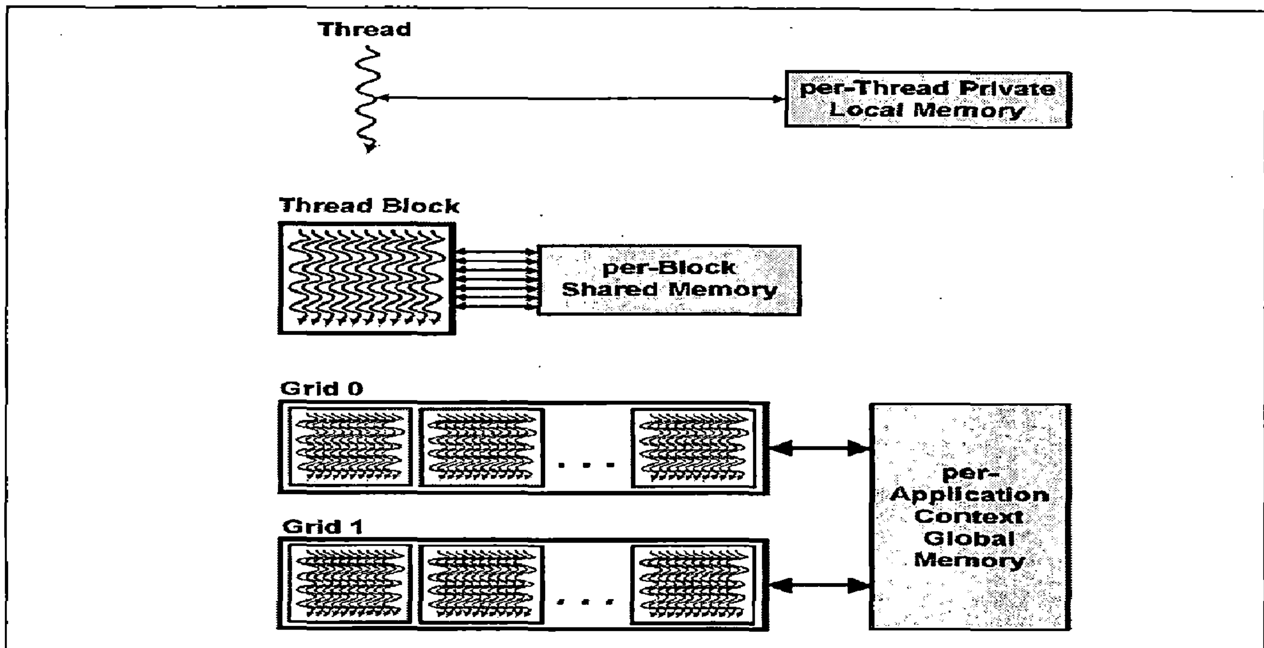



**Figure 2.4 Memory Hierarchy in CUDA programming model**

Each thread has a private local memory. Each thread block has a shared memory visible to all threads of the block and with the same lifetime as the block. Finally, all threads have access to the same global memory [11].

The goal of CUDA programming model is to make GPU programming simple for users familiar with C programming. It provides a rich API Library consisting of functions that can be used to program the GPU. Using CUDA it is possible to access the GPUs for computation as CPUs. Using the CUDA programming model the programmer need not worry about the device (GPU) details and just concentrate on developing his application. It also provides a runtime library providing functions to control one or more devices (GPU) from the host (CPU), device specific functions, and some built-in vector types supported on both host and device. Figure 2.5 gives the details of the CUDA software stack [11].

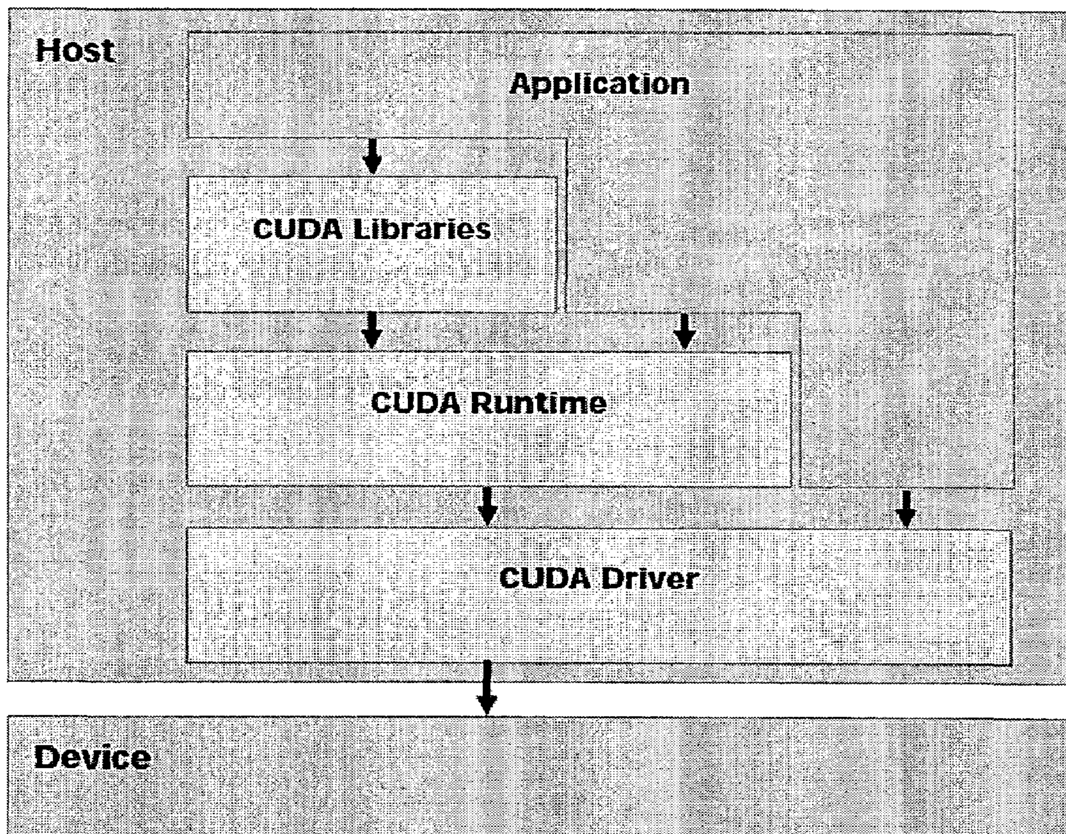



**Figure 2.5 CUDA Software Stack**

## 2.6 Literature Review

This section gives the literature review of parallelisation of text classification algorithms using CUDA, parallelisation of Naïve Bayes' classification algorithm and the use of semantic information in text representation.

### 2.6.1 Parallelisation of Text Classification algorithms using CUDA

One of the first works towards using GPUs for Text Classification was done by Y. Zhang et al [12]. They proposed a parallel implementation of the TFIDF text mining algorithm utilising the massive compute capabilities of the GPUs. They parallelised the various modules involved in the task of text classification ranging from tokenization, stemming, document hash table creation and calculation of TFIDF. They used the CPU/GPU

heterogeneous model where the CPU acts as the core processor and the GPU acts as the compute accelerator. They developed their algorithm such that when the GPU is busy developing the document hashtable the CPU can load the next batch of input, thus preventing any of the processor from sitting idle. They published the results comparing the time required for GPU implementation of each module against the time required for CPU implementation by varying the corpus size. They also gave a few drawbacks of their implementation one of which is the non-coalesced memory accesses. Their results showed that GPU implementation is up to six times faster than the CPU implementation.

S. Liang et al [13] proposed a parallel implementation of the *k-nearest neighbours* classification algorithm using CUDA. Their implementation consists of two GPU kernels: distance calculation kernel and sorting kernel. It is the distance calculation kernel which majorly affected the execution time of the algorithm. The distance calculation kernel maximizes the concurrency of the distance calculation between various threads. They also made a good use of the shared memory of each core on the GPU. The training dataset was loaded into the shared memory of each core and the threads on the same block shared the training data with each other. They also applied optimization strategies like pipelining and coalesced memory accesses. Once the distances between the training objects and the unknown object *p* are available, the sorting kernel was used to find the k nearest neighbours to *p*. The results published, show that speedup of up to 15X was achieved over the CPU implementation.

### 2.6.2 Parallelisation of Naïve Bayes' classification algorithm

C. Kruengkrai and C. Jaruskulchai [14] proposed a parallel learning algorithm for text classification that made use of the Naive Bayes' algorithm. They combined the learning step of Naive Bayes with the Expectation-Maximisation (EM) algorithm to handle the unavailability of a large set of labelled training documents. But since the EM algorithm does not scale well and slows down as the dataset size grows, they made use of parallel processing. Their experiments were performed on PIRUN Cluster at Kasetsart University. PIRUN Cluster consists of 72 nodes connected with Fast Ethernet Switch 3COM SuperStackII. Each node is a 500 MHz Pentium III with 128 Mbytes of RAM and uses Linux as the operating system. Each processor in the system ran its own copy of the same

program thereby making use of the Single Program Multiple Data (SPMD) paradigm for parallelisation. 20 Newsgroups dataset was used in their experiments. Their results show that the parallel implementation with 16 processors was up to 12 times faster than the single CPU implementation. Moreover, they also showed that as the number of parallel processors decrease the speedup achieved also decreases.

W. Ding et al [15] proposed a new Naive Bayesian text classifier, Package and Combined Naive Bayesian classifier (PC-NB), which relaxed the independence assumption without compromising on the efficiency. They also parallelised the prediction step of the existing Naive Bayes algorithm and their proposed algorithm on a cluster of FANGZHENG PC (Memory: 256M, CPU: 1.6Hz) computers. They used the MPI model for parallel programming. Reuters-21578 and the Industry Sector4 datasets were used for performance comparisons. The results show that as the parallelism is increased by increasing the number of computation nodes, the time required by the naive bayes algorithm decreases before saturating beyond which no decrease in time could be observed even on increasing the computation nodes.

V. G. Roncero et al [16] also proposed a parallel learning algorithm for text classification using the combination of naive bayes and EM as in [14]. But instead of using a cluster as the hardware for parallelisation they made use of the grid environment. The use of grid environment allows for distributed collection of training data and distributed processing. Their paper does not give any experimental results showing the improvements obtained by using the grid environment for text classification.

### 2.6.3 Text representation using semantics

One of the first works towards using WordNet for capturing the semantic information in text was published by Rodriguez et al. [17]. They focussed on using WordNet to enhance neural network learning algorithms to improve the classification accuracy on Reuters 21578 corpus. They also retained the bag-of-words representation of text. However, their approach only makes use of the synonymy. Their approach also took advantage of the fact that the Reuters topic headings are themselves good indicators for classification. This makes their work biased towards the Reuters-21578 dataset and may not work with

datasets where their assumption is false.

S. Scott, S. Matwin [18] extended the work of Rodriguez to make use of POS tagging. But they also modified the representation of text to make use of English language phrases as features instead of the bag-of-words representation. Specifically they used the part of speech information from the Brill tagger and the synonymy and hypernymy relations from WordNet to change the representation of documents to hypernym density.

L.S. Jensen, T. Martinez [19] tested the method suggested by Rodriguez and Scott along with other methods on 3 different datasets: A subset of the Reuters-21578 data; a collection of USENET postings; and a repository of folk songs called the Digital Tradition. They used 3 different classification algorithms in their work, coordinate matching, TF*IDF, and naive Bayes. Their results show that the use of synonyms does increase the classification accuracy and can be used easily for tasks where a high accuracy is required.

## 2.7 Research Gaps

Based on the literature review the following research gaps have been identified.

- There is need to parallelise the Naïve Bayes' text classification algorithm because the serial algorithm is slow and the vast amount of information in the form of text needs to be processed as fast as possible.
- The Naïve Bayes' algorithm needs to be parallelised using the GPUs as GPUs offer a better price to performance gain ratio as compared to other approaches of parallelisation like grid computing, cluster computing etc.
- With the increase in the number of predefined classes in the task of classification, the existing algorithms become slower in classifying a text document to one of the predefined categories. Thus there is a need to develop an algorithm, such that the amount of time required for classifying a given text document does not increase with the increase in the number of classes.

# Chapter 3

# Proposed Framework

Figure 3.1 gives the architecture of the parallel Naïve Bayes' algorithm for classification of unstructured text documents.

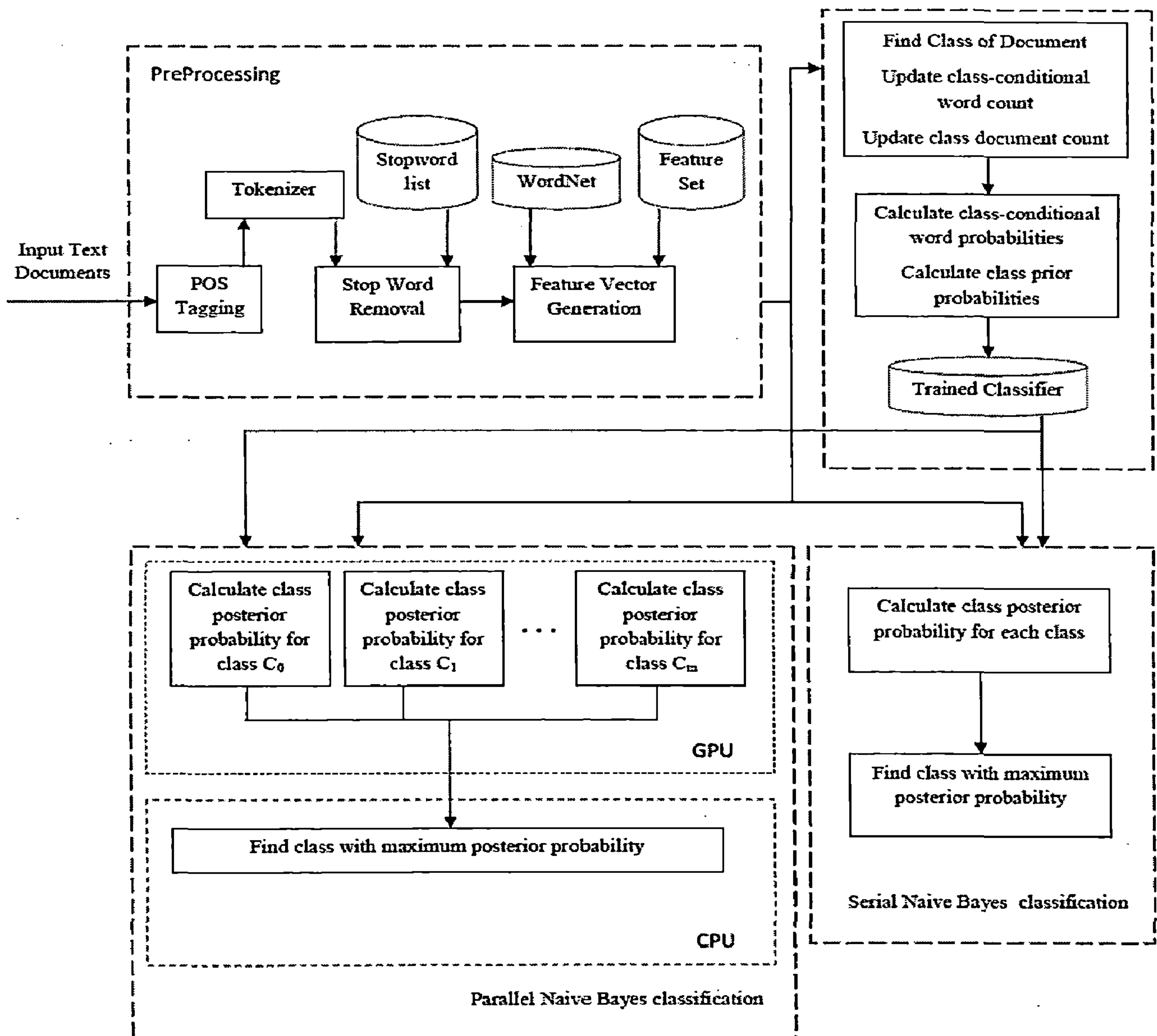



**Figure 3.1 System architecture of Naive Bayes' text classification**

A brief description of the various modules involved is given in the following sub-sections. The detailed description has been given in the next chapter.

## 3.1 Preprocessing Module

The first and foremost step is the conversion of a text document to a feature vector representation. This is done in the preprocessing module. The module can be divided into smaller modules like POS tagging, tokenizing, stop word removal, and then finally converting the document to a feature vector.

The given text document is passed through the POS tagger, the output of which is a string, each word in which is annotated with its part-of-speech. This string is then tokenized into a list of tokens. Each token is a word/part-of-speech pair. The word is then provided as input to the sub-module for stop word removal. If the word is not a stop word, then the token is given as input to the feature vector generation sub-module. The module uses the information from the WordNet database and the feature-set to generate the feature vector for the document. The feature set used has been constructed using the training dataset.

## 3.2 Training Module

The training dataset comprising of text documents in the form of feature vectors, is provided as input to the training module. The module processes the feature vector of each document one by one through a series of steps. The first step in the module is to determine the class of the input document and increase the document count for that class. Then it updates the class-conditional word counts for the corresponding class utilizing the feature vector of the document.

Once all the training documents have been processed, the class-conditional word probabilities are calculated. The class prior probabilities are also calculated and then the

results are written to disk so that the classifier can be used later and there is no need to train the classifier whenever a text document needs to be classified.

## 3.3 Serial Naïve Bayes Classification

To compare the results of the parallel Naïve Bayes' classification algorithm, the serial Naïve Bayes' algorithm was used. The input to the serial Naïve Bayes' classification module is the feature vector of the document to be classified and the trained classifier. The module calculates the posterior probability of each class for the given document in a serial fashion. The calculation of the posterior class probability involves the multiplication of all the class-conditional word probabilities for this class. Once the probability of each class has been calculated the next step is determine class with the maximum probability.

## 3.4 Parallel Naïve Bayes' Classification

This module parallelizes the calculation of posterior class probabilities given a text document. The module takes the feature vector of the document and the trained classifier as input and gives the class of the document as the output.

Instead of calculating the probability of each class in a serial fashion, the calculations for each class are done simultaneously on a GPU. The detailed design and implementation of the module has been given in the next chapter.

# Chapter 4

# Detailed Design and Implementation

This chapter gives the detailed design and implementation of the proposed parallel Naïve Bayes' text classification. As discussed in the previous chapter, the task of Naïve Bayes' text classification could be divided into three major modules which are discussed in the following sections.

## 4.1 Preprocessing Module

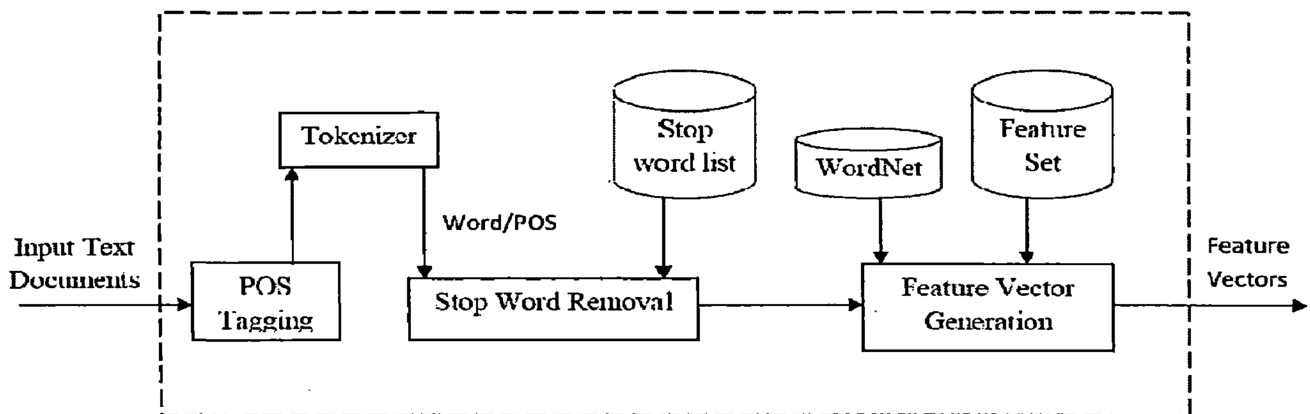



**Figure 4.1 Block Diagram of pre-processing module**

The input to the system comprises of documents containing unstructured text. These documents are converted into a feature vector in a series of steps as shown in Figure 4.1. Maxent POS Tagger is used for POS tagging. Stop word removal is done with the help of a stop list which is provided as input to the system. Semantic information from WordNet is used to finally convert the list of tokens (words) into the feature vector representation. The feature set to be used is provided as input to this sub-module. Figure 4.2 gives a detailed flowchart of the steps involved for a given text document.

M.G. CENTRAL LIBRARY
ACC No..............
Date..................
I.I.T. ROORKEE

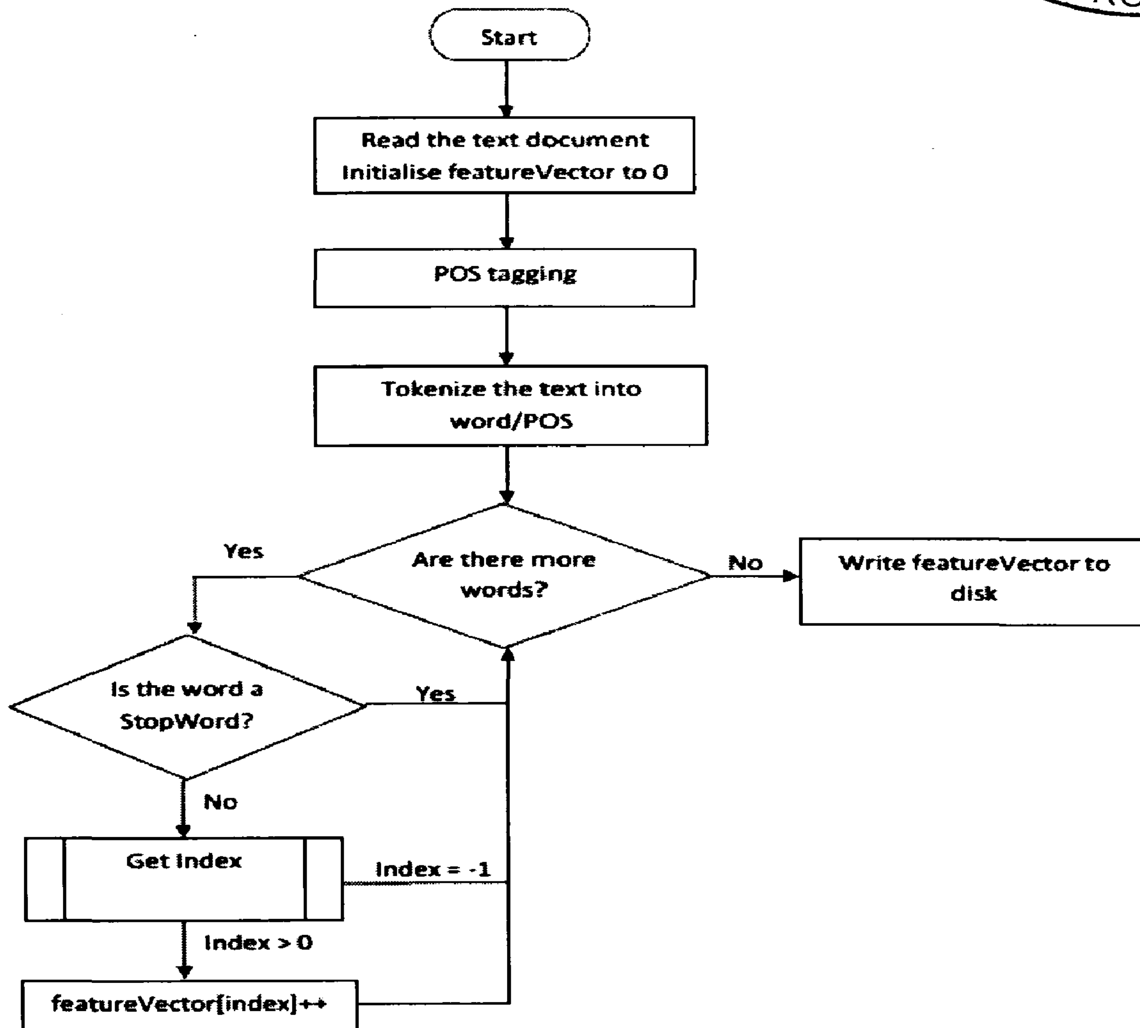



**Figure 4.2 Flow Chart of the pre-processing module using POS tagging and WordNet synsets**

The process begins by reading the input text document, which is then passed through the POS tagger. The POS tagger tags each word (term) in the document with its proper part-of-speech. This tagged string of the document is then tokenized into *word/POS* tokens. Each word is then passed through the stop word removal module after which the token is passed to the *GetIndex* subroutine. The subroutine *GetIndex* has been described in Figure 4.3. The routine returns the index of a feature matching the current *word/POS* token or a -1. If it returns -1, it means that the token is not in the feature-set and not is used for representing the document. After each token has been processed the feature vector of the document is available for use in other modules.

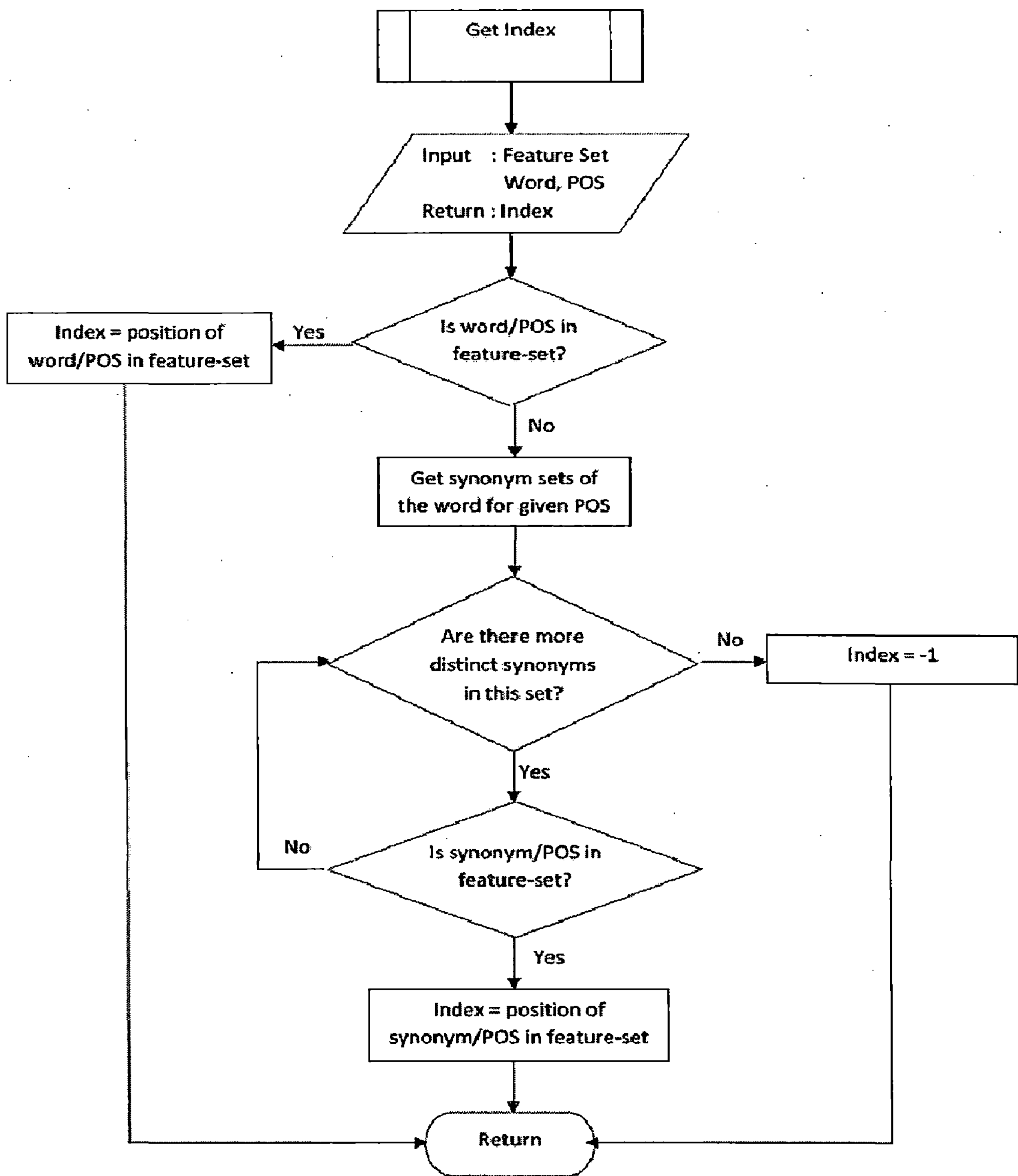



**Figure 4.3 Flow Chart of the Get Index sub-routine using POS tagging and WordNet synsets**

The feature set used in the system could either be provided by human input or constructed automatically using the training dataset. In the proposed parallel Naïve Bayes' text classification, the feature-set has been constructed automatically using the training dataset. The steps involved in the generation of feature-set are very similar to those outlined in Figure 4.2 and Figure 4.3. The only difference is that when the *GetIndex* subroutine returns a -1, signifying absence of the token from the feature-set, that token is added to the feature-set. And when the return value is positive, signifying presence of the token in the feature-set, next token is chosen for processing from the list of tokens.

The preprocessing module was also implemented without using the POS tagging sub-module. Since the POS tagging was not used, WordNet synsets corresponding to all possible uses of the word are retrieved. As a result, it may be possible that unrelated synonyms are merged together to point to the same feature resulting in a drop in some accuracy.

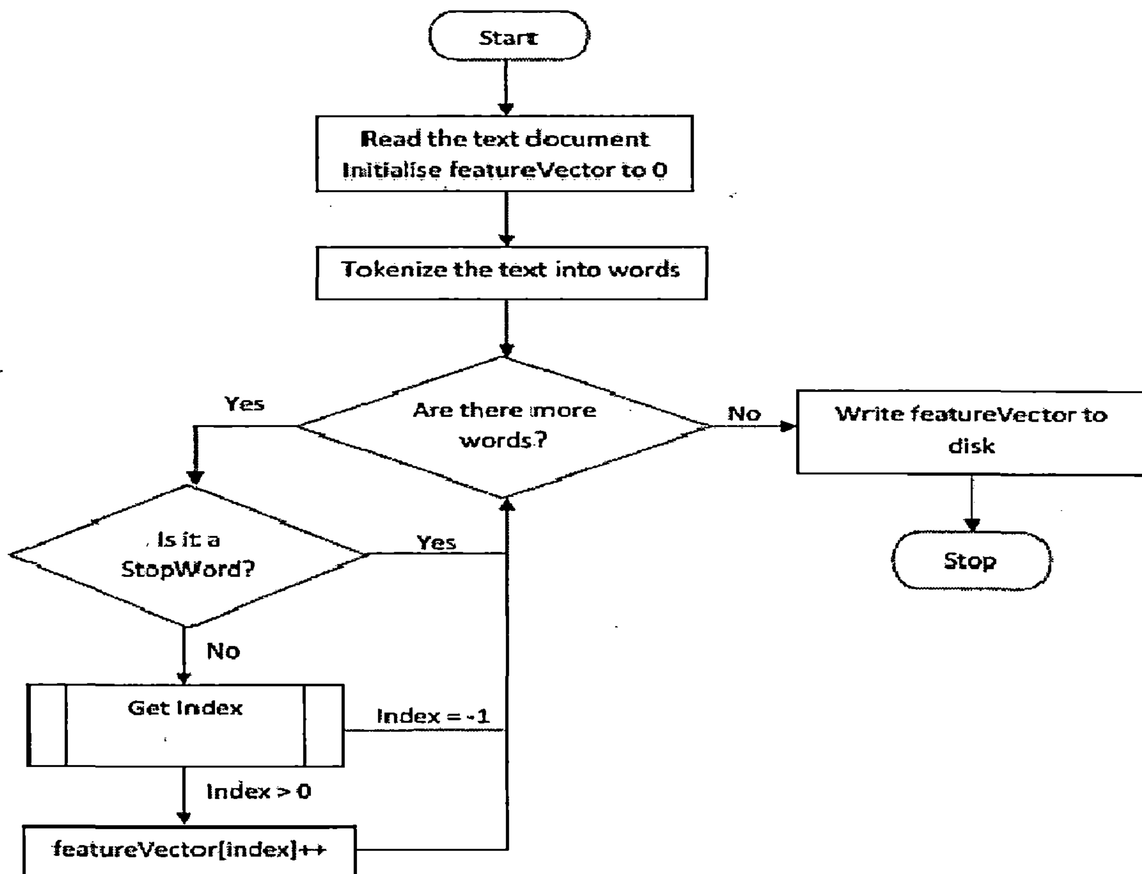



**Figure 4.4 Flow Chart of Pre-processing Module using WordNet synsets**

The flowchart in Figure 4.4 gives the outline of the process involved in feature vector generation using WordNet. The corresponding *GetIndex* subroutine is also modified as shown in Figure 4.5.

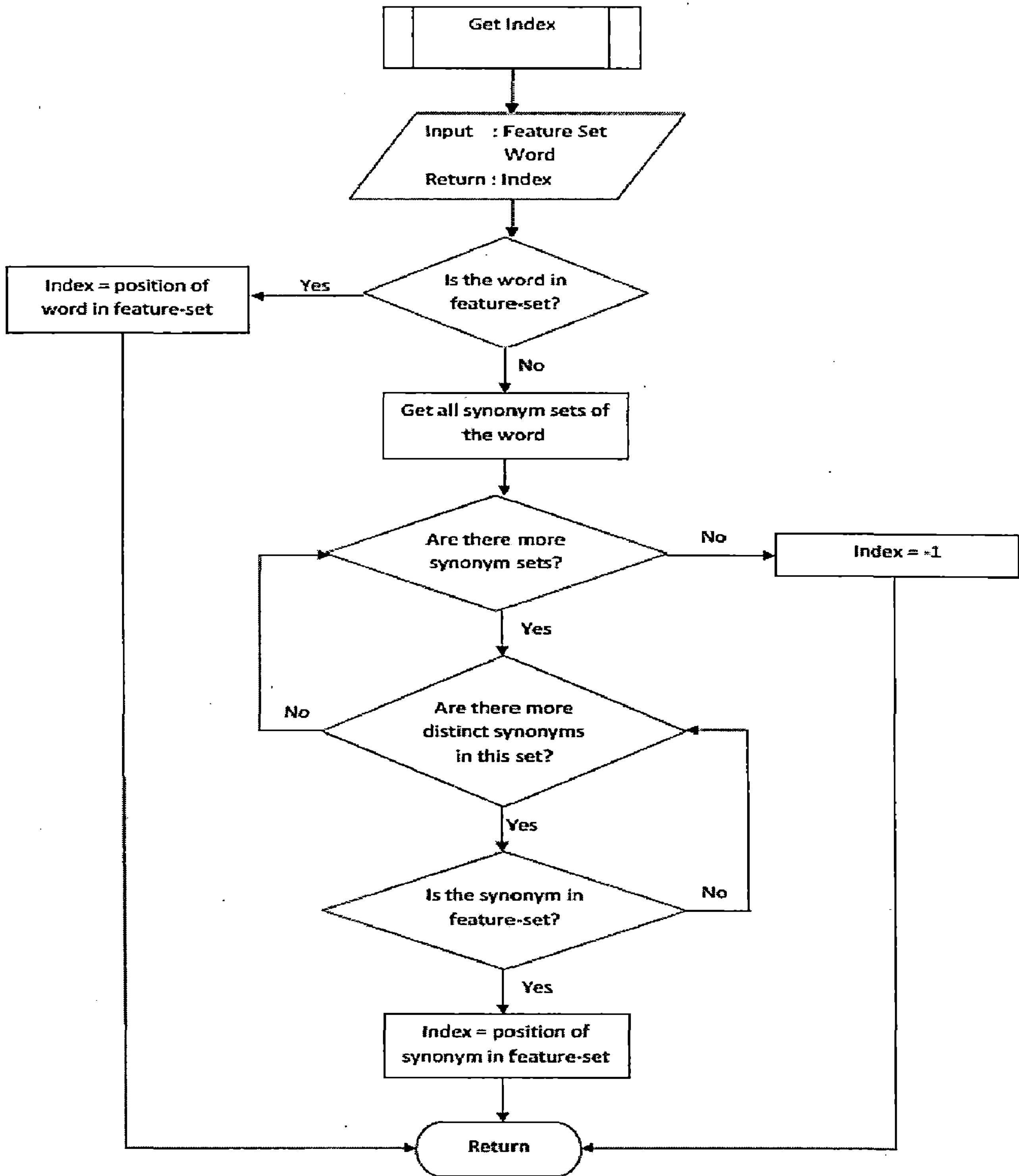



**Figure 4.5 Flow Chart of the Get Index sub-routine using WordNet synsets**

For comparison purposes the preprocessing module was also implemented without making use of any semantic information of the given text documents. In this approach the steps involved are exactly those as given in Figure 4.4. However the *GetIndex* subroutine just checks if the given word is in the feature-set and returns its index in the same. However, if the word is not present in the feature-set it returns -1.

## 4.2 Training Module

The steps involved in training the classifier have already been outlined in the proposed framework. Figure 4.6 gives the detailed flow diagram of the steps involved in training the Naïve Bayes' classifier.

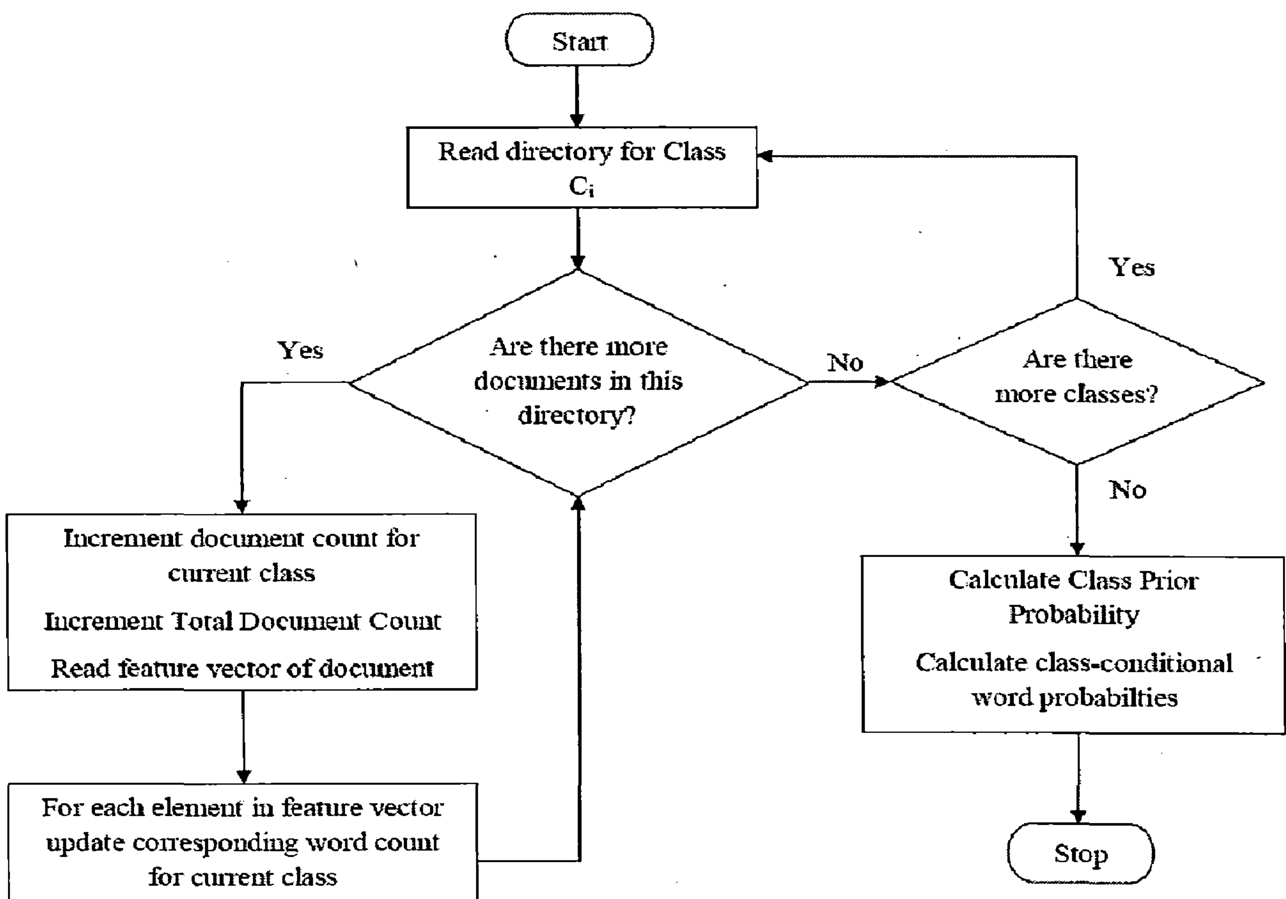



**Figure 4.6 Flow Chart of the Training Module of Naive Bayes' text classification**

The multinomial Naïve Bayes' model discussed in section 2.1.2 has been used in the implementation of this module. The module takes as input the training dataset consisting of text documents in the form of feature vectors. The module requires that the documents be grouped class-wise such that each class is represented by a directory and contains all the documents belonging to that class. All these directories are present in a single root directory which is provided as input to the module. After building the classifier, i.e., calculating the class-conditional word probabilities and the class prior probabilities, the module writes them to disk so that the classifier is available for use later. The classifier built is tested on the test dataset using the classification module which is described in the next section.

## 4.3 Classification Module

The classification step of the Naïve Bayes' model involves the calculation of posterior probability of each class given the test document and then finding the class with maximum probability. As already mentioned, the multinomial Naïve Bayes' model has been used to implement the proposed framework. Using the equations in section 2.1.2, the relationship given by Equation 4.1, can be deduced for the posterior class probability conditional to the document.

$$P(c_j \mid d_i) \propto P(c_j) \prod_{t=1}^{|V|} \frac{P(w_t \mid c_j)^{N_{it}}}{N_{it}!} \quad (4.1)$$

The terms like $P(d)$ made no changes to the decision as for a given document they remained constant even when the class is changed, and were hence removed. The term $N_{it}!$ could also be removed as the product $N_{i0}! * N_{i1}! * N_{i2}! * \cdots * N_{i|V|}!$ is constant over the range of classes for a given document $d_i$. Hence the relationship reduces to the one given in Equation 4.2

$$P(c_j \mid d_i) \propto P(c_j) \prod_{t=1}^{|V|} P(w_t \mid c_j)^{N_{it}} \quad (4.2)$$

The complexity of the algorithm is of the order of $O(mn)$ where $m$ is the number of classes and $n$ is the number of features.

The Naïve Bayes' classification module has been parallelised such that the calculation of the posterior class probability $P(c_j|d_i)$ of each class is performed simultaneously. The calculations for $P(c_j|d_i)$, i.e., the multiplication by $P(w_t\,|\,c_j)^{N_{it}}$, are also parallelised such that each thread processes a part of the feature vector and calculates a partial product. Figure 4.7 gives an outline of the model of the parallel Naïve Bayes' classification module.

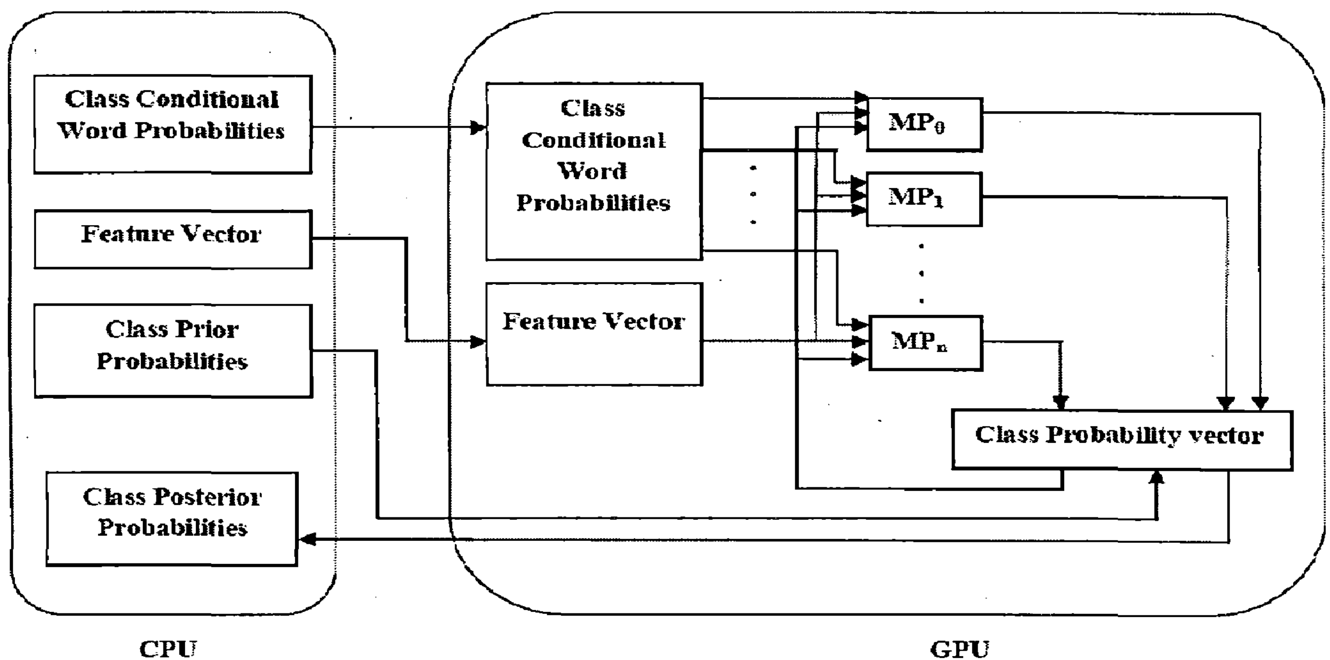



**Figure 4.7 Model for parallel Naive Bayes' text classification module**

The class-conditional word probability matrix and the class prior probabilities first need to be transferred to the GPU memory. The feature vector of the unknown document also needs to be transferred to the GPU memory. Each block accesses the entire feature vector but only a row of the class-conditional word probability matrix. The class-conditional word probability is stored as an $m$ x $n$ matrix where $m$ is the number of classes and $n$ is the number of features. Figure 4.8 gives the details of the data structures used for representing the class-conditional word probabilities, feature vectors and the class prior probabilities. Figure 4.9 gives the details about the processing done by a multiprocessor.

| | $W_0$ | $W_1$ | ... | $W_n$ |
|---|---|---|---|---|
| $C_0$ | $P_{00}$ | $P_{01}$ | ... | $P_{0n}$ |
| $C_1$ | $P_{10}$ | $P_{11}$ | ... | $P_{1n}$ |
| . | . | . | . | . |
| . | . | . | . . . | . |
| . | . | . | . | . |
| $C_m$ | $P_{m0}$ | $P_{m1}$ | ... | $P_{mn}$ |

Class-conditional word probability matrix

| | $W_0$ | $W_1$ | ... | $W_n$ |
|---|---|---|---|---|
| $D_i$ | $tf_{i0}$ | $tf_{i1}$ | .... | $tf_{in}$ |

Feature Vector

| $C_0$ | $C_1$ | ... | $C_m$ |
|---|---|---|---|
| $P_0$ | $P_1$ | ... | $P_m$ |

Class Probability Vector

$P_{00}$ : Probability of $word_0$ given $class_0$

$tf_{i0}$ : frequency of $word_0$ in $document_i$

**Figure 4.8 Representation of class-conditional word probability, feature vector and class probability**

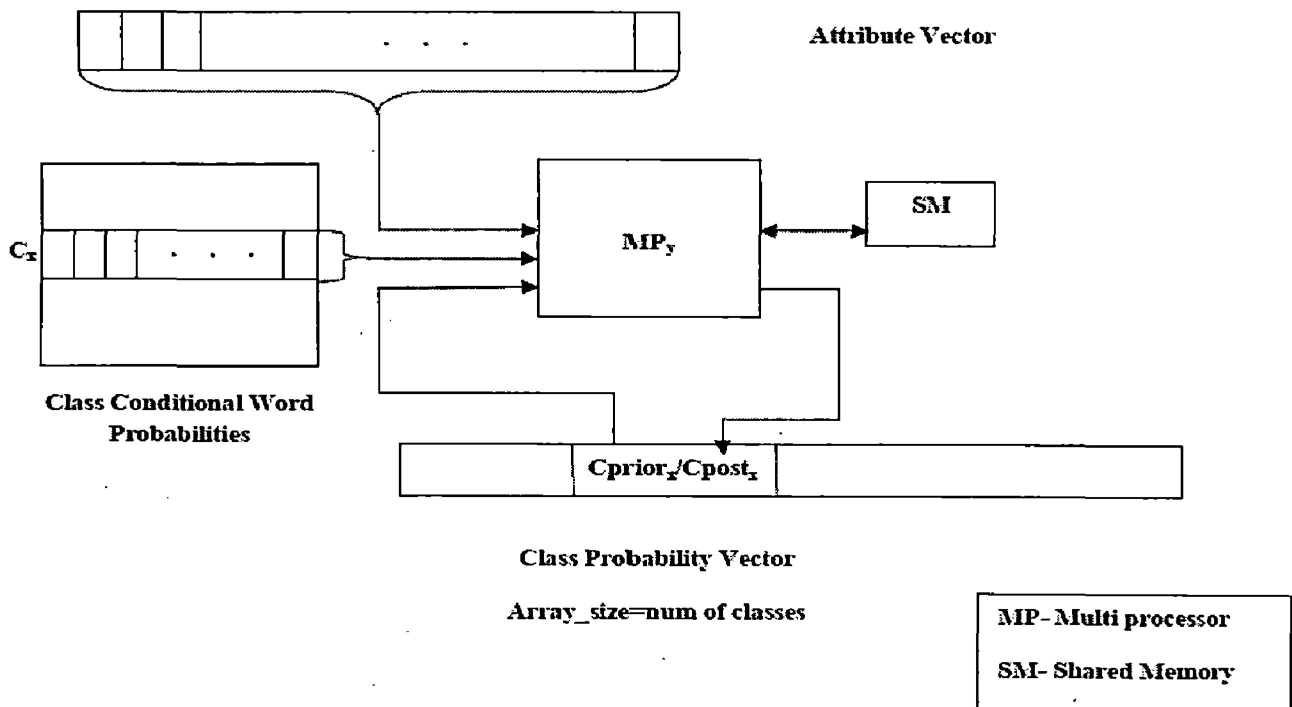



**Figure 4.9 Processing done by a block (or MP)**

The block calculates the posterior probability for the class corresponding to the row of the matrix accessed. This also requires the number of blocks to be equal to the number of classes under consideration. More number of classes leads to more number of blocks and hence more parallelism. A block is executed on a multi-processor (MP). It may so happen that a MP executes more than one block, but it will never be the case that a block is split between 2 or more MPs. The class probability vector is used as an input/output vector. Initially it stores the class prior probabilities, then the temporary results during the execution of the kernel and then the final result which gives the posterior class probabilities. This result is then returned to the CPU. Since each block accesses only one element of the class probability vector, to speed up the processing, local variable which gets stored in a register, is used for storing the temporary results.

A number of threads are created on each block. Each thread processes some part of the feature vector which is divided equally among them. The number of threads that are created on each block is a multiple of 32. This is required because each MP consists of 8 SP and the fastest instruction takes 4 cycles. Therefore each SP can have 4 instructions in its pipeline for a total of 8*4 instructions being executed concurrently on a MP. Within a warp (number of threads executing concurrently on a MP) threads have sequential indices. Thus there is a warp with thread indices 0...31, another with 32...63 and so on. The homogeneity of the threads used, makes it possible for all SPs on the MP to execute the same instruction in parallel.

Since the number of threads is a multiple of 32 whereas the number of features may not be, some elements of the feature vector and a row of the class-conditional word probability matrix are not assigned to any thread. These remaining elements are processed by threads $T_o - T_x$ ($x <$ number of threads), once these threads have finished processing their batch of input. This leads to some amount of performance degradation due to serialization. To avoid this, efforts must be made to keep the number of features a multiple of 32.

Two global memory access strategies have been used in the parallelization of Naïve Bayes. First, in which the vector was divided into $k$ groups, $k$ being the number of threads on the block, and each thread accessed all the elements of a group. Since the addresses

generated by the threads in a warp are not sequential but are separated, multiple memory accesses are required as the GPU cannot coalesce them into a single access. This slows down the module. In the second strategy, the memory accesses are such that the threads in a warp access sequential memory addresses. Figure 4.10 will make things clear.

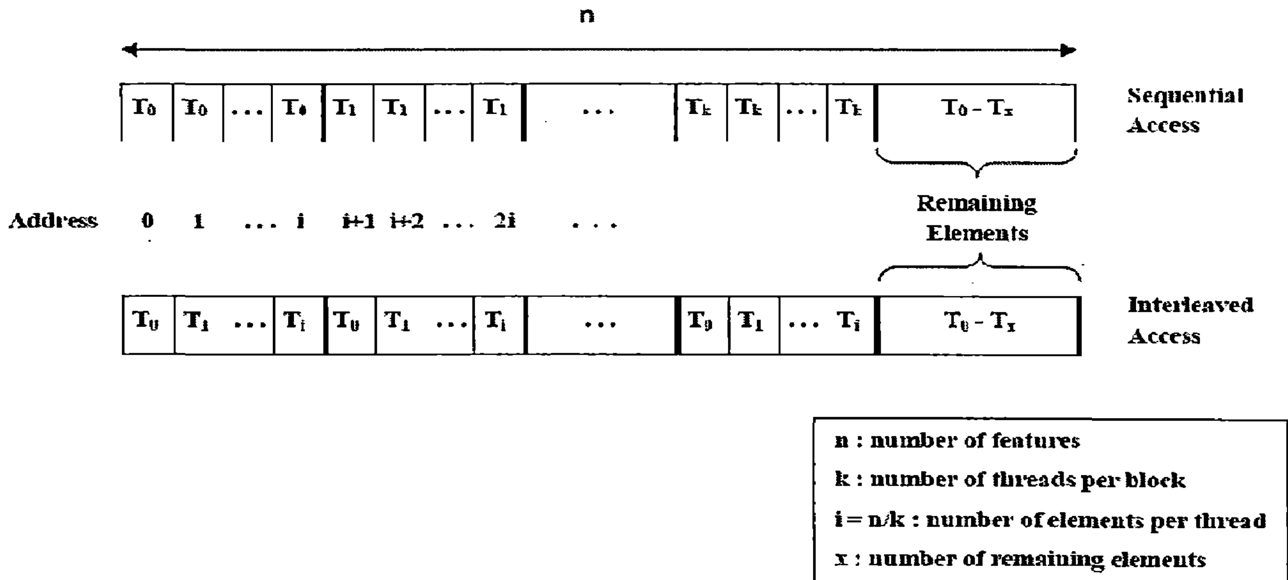



**Figure 4.10 Memory Access Strategies for feature vector and class-conditional word probabilities**

Figure 4.11 gives the addresses generated by the threads in a warp for the two memory access strategies discussed.

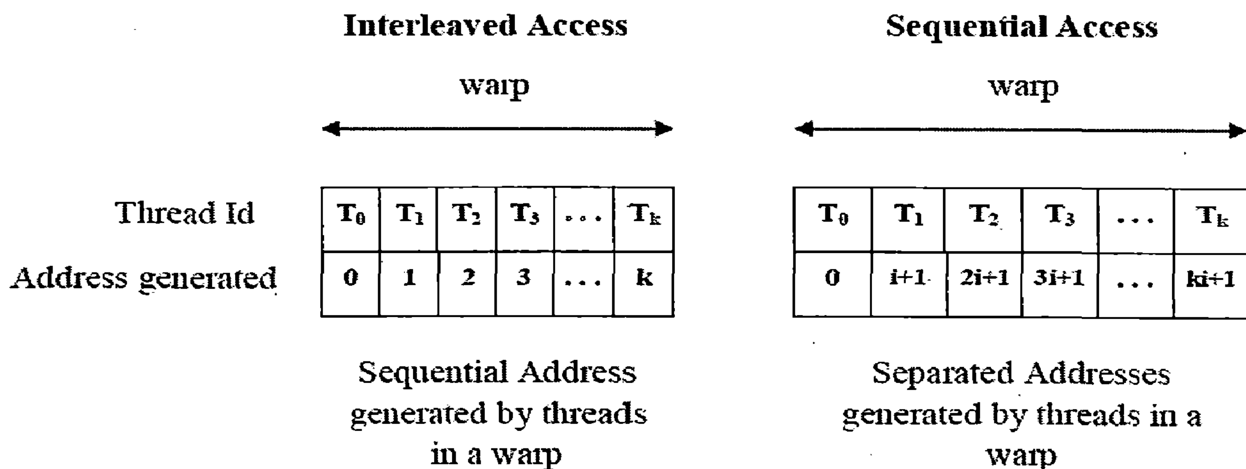



**Figure 4.11 Addresses generated by threads in a warp**

Once each thread is done with calculating the partial products, their results are written to a vector in shared memory. Shared memory is much faster than global memory and is shared by all threads on the block. Each block has its own shared memory. The shared memory is limited and must be usedjudiciously.

The shared memory was also used to store the input class-conditional word probabilities and the feature vector. But no improvement was observed over the parallel implementation without using shared memory. Instead there was a performance penalty when shared memory was used for storing the input data over the GPU. The reason for this performance degradation is: In parallel Naïve Bayes' classification, an element in the class-conditional word probability and the feature vector is accessed only once inside each block. With the use of shared memory the memory accesses increase as the data has to be first read from global memory and written to shared memory and then read from shared memory and used in computations. Whereas, if global memory is used the data is read from global memory and directly used in computations. Thus, one global memory access has to be present. But with the use of shared memory, an additional memory read (write) from (to) the shared memory gets involved, which is a performance overhead.

The partial products obtained from each thread need to multiplied together to obtain the final class probability. Instead of using one thread to accomplish this, multiple threads are used in a parallel manner as shown in Figure 4.12.

The first step involves utilizing the first *k/2* threads of the block and each thread multiplies the values at indices given by *'threadId'* and *'threadId + k/2'* and stores the result at index *'threadId'*. After this step we are left with a total of *k/2* values to be multiplied together to obtain the final result. In a similar fashion the second step involves utilizing *k/4* threads to obtain *k/4* values and this goes on until we get a single value. After the first step of utilising *k/2* threads we wait till each thread has done its processing because the output of this step forms the input of next step.

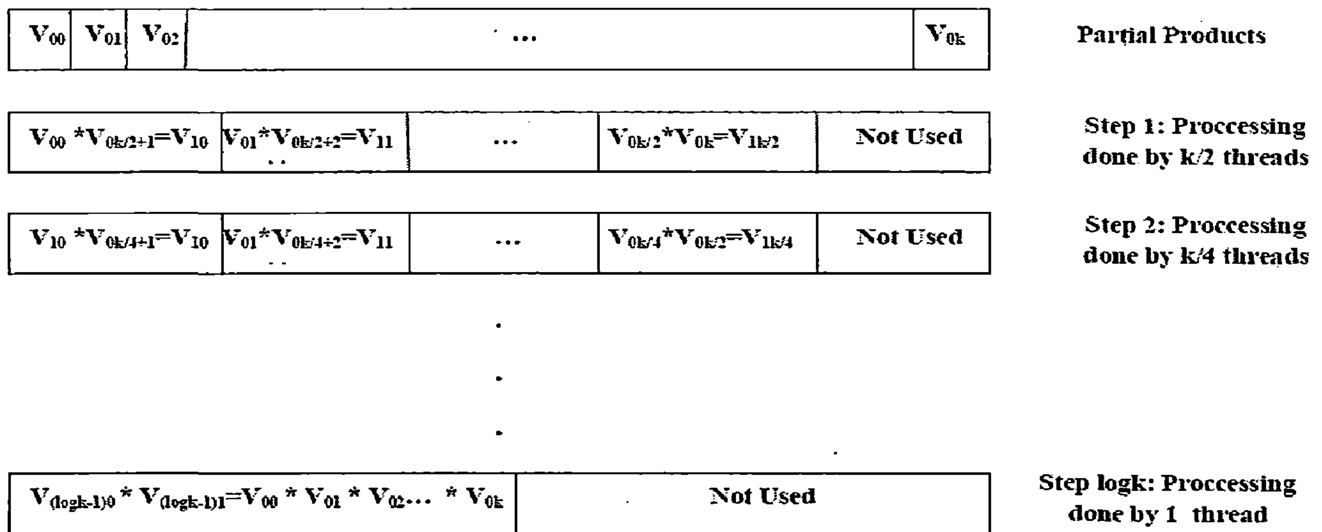



**Figure 4.12 Processing done to obtain final class probability**

The final value is then written to the class probability vector at the location corresponding to this block which also corresponds to a class. After all the blocks have completed their tasks, the posterior probability of each class is available in the vector. Then the maximum value is searched in the vector and the corresponding class is assigned as the class of the unknown document. Since this step is not computation intensive we let the CPU perform this task.

## 4.4 Experimental Setup

This section describes the dataset used for testing the parallel Naïve Bayes' text classification algorithm. The CPU and GPU hardware used for the experiments have also been discussed in this section.

### 4.4.1 Data Preparation

Reuters 21578 dataset has been used to test the parallel naïve bayes algorithm. It is a collection of news articles that appeared on the Reuters newswire. The dataset has been the most widely used one for research on text classification.

The dataset is organized into 21 files each having 1000 articles and the remaining are in the 22nd file. Each article in the file has been tagged with information like *title, category, article number, train/test document* and many more.

To prepare the input data, each article is written to a separate file whose name is given by *article number.* Also the articles belonging to the same *category* are grouped into a single directory. The train/test documents are also separated by using the information provided.

We also order the categories (classes) according to the number of documents in them and then group them into 16 most frequent categories, 32 most frequent categories and so on.

### 4.4.2 Hardware Configuration

The GPU used for testing the parallel naïve bayes text classification algorithm was Nvidia GTS 250. Table 4.1 gives the details about the graphics card. To compare the results, Pentium P4 processor was used to run an efficient CPU implementation of Naïve Bayes', the specifications of which are given in Table 4.2.

**Table 4.1 Hardware Specifications of GTS 250 card**

| | |
|---|---|
| CUDA Cores | 128 |
| Graphics Clock | 738 MHz |
| Processor Clock | 1836 MHz |
| Memory Clock | 1100 MHz |
| Memory | 512 MB of GDDR3 RAM |
| Memory Interface Width | 256-bit |
| Memory Bandwidth | 70.4 GB/s |

**Table 4.2 Hardware Specifications of the CPU used**

| # Cores | 2 |
|---|---|
| Clock Speed | 3 GHz |
| Cache | 2 MB |
| FSB Speed | 800 MHz |
| Memory | 4 GB |

# Chapter 5
# Results and Discussions

The results for the proposed methodology and the experimental setups have been explained in this chapter.

## 5.1 Comparison of classification accuracy

To check if the implementation of Naïve Bayes' was correct or not, its classification accuracy was compared to that given by Weka on the same dataset. Table 5.1 gives the results of the two techniques on the datasets formed by selecting documents from 16 and 32 most frequent classes of the reuters-21578 dataset respectively.

**Table 5.1 Comparison of Classification Accuracy of proposed parallel Naïve Bayes' text classification with that of WEKA**

| Dataset | Classification Accuracy (%) | |
|---|---|---|
| | Weka | Naïve Bayes' |
| 16 most frequent classes of reuters-21578 | 69 | 68.8 |
| 32 most frequent classes of reuters-21578 | 68.47 | 68.17 |

As is clear from Table 5.1, the results of the implementation presented in this thesis are the same as that produced by Weka. Thus it can be said that the implementation is correct. The results of using WordNet and POS tagging in the preprocessing module are discussed in the next sub-section.

### 5.1.1 Results of integrating WordNet and POS tagging

To use the semantic information present in the given text documents for improving the quality of classification WordNet and POS tagging were used in the preprocessing module of the proposed parallel Naïve Bayes' text classification algorithm. Table 5.2

below gives the results obtained. Two different sets of test data were used. Both the datasets are a subset of the Reuters-21578 dataset.

**Table 5.2 Comparison of Classification Accuracy obtained by using WordNet and POS tagging**

| Dataset | Classification Accuracy (%) | | |
|---|---|---|---|
| | Naïve Bayes' | Naïve Bayes' + WordNet | Naïve Bayes' + WordNet + POS Tagging |
| 16 most frequent classes of reuters-21578 | 68.8 | 72.2 | 73.97 |
| 32 most frequent classes of reuters-21578 | 68.17 | 71.5 | 73.49 |

The results show that the use of WordNet and POS tagging even in the simplest of the ways leads to an improvement in the classification accuracy. But the improvements observed are not that great suggesting the use of some intermediate steps if highly increased classification accuracy is desired.

## 5.2 Results of Parallelization

The results of the GPU implementation of naïve bayes text classification are presented in this section.

### 5.2.1 Determining the number of threads

To determine the number of threads/block to be made, the implementation was run for different values keeping the number of blocks (= number of classes) constant at 112. The experiments were made using 32, 64, 128, 256, and 512 threads on each block. Table 5.3

gives the execution time required by both the memory access approaches (sequential and interleaved). Figure 5.1 gives the plot of the results.

**Table 5.3 Comparison of Execution Time of the two parallel Naïve Bayes' implementations on GPU by varying number of threads/block**

| Number of threads | Execution Time (ms) | |
|---|---|---|
| | Sequential Access | Interleaved Access |
| 32 | 3.71 | 1.47 |
| 64 | 2.67 | 1.45 |
| 128 | 2.47 | 1.43 |
| 256 | 2.73 | 1.46 |
| 512 | 4.28 | 1.48 |

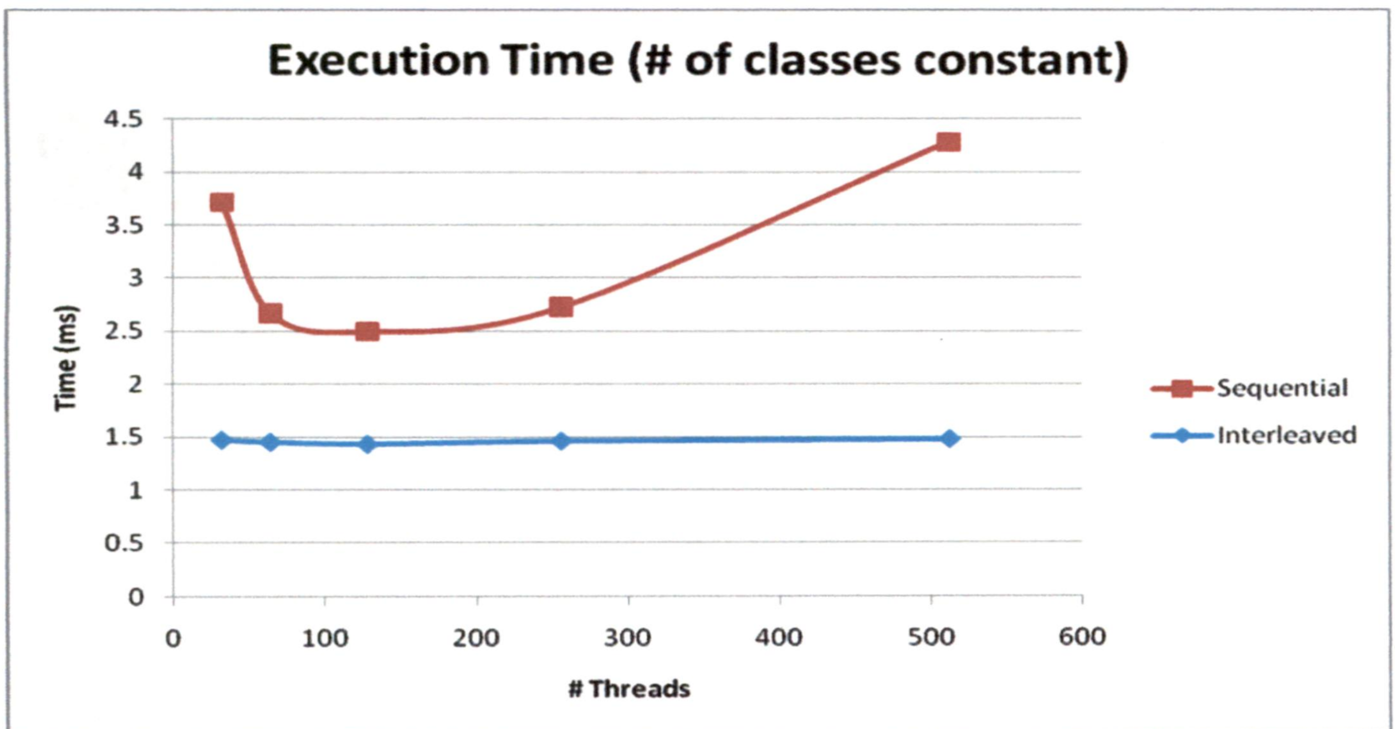



**Figure 5.1 Comparison of Execution Time of the two parallel Naïve Bayes' implementations on GPU by varying number of threads/block**

As is clear from Table 5.3 and Figure 5.1, the thread value of 128/block requires the least amount of time as compared to the other values. Thus this value of threads/block is chosen for the experimental setup. The execution time listed in Table 5.3 does not take into consideration the time required to transfer the necessary data to and from the GPU. This is not required because both the approaches will require the same amount of data transfer time and we are interested only in their relative time requirements.

### 5.2.2 Comparison of parallel Naïve Bayes' with serial Naïve Bayes'

In Table 5.4, the execution time of CPU implementation of Naïve Bayes', parallel Naïve Bayes' using sequential memory accesses and parallel Naïve Bayes' using interleaved (coalesced) memory accesses on GPU, has been given. The GPU time is the summation of the time required for data transfer from CPU memory to GPU memory and vice-versa, and the time required for calculating the posterior class probability. The CPU time is just the time required for calculation of posterior class probability.

**Table 5.4 Comparison of Execution Time by varying number of classes**

| Number of classes | Execution Time (ms) | | | Speed Up (vs. CPU) CPU Time / GPU Time | |
|---|---|---|---|---|---|
| | CPU | GPU (sequential Access) | GPU (coalesced access) | GPU (sequential access) | GPU (coalesced access) |
| 16 | 35.97 | 1.36 | 1.22 | 26.45 | 29.48 |
| 32 | 75.44 | 2.47 | 2.01 | 30.54 | 37.53 |
| 48 | 115.88 | 3.39 | 2.77 | 34.18 | 41.83 |
| 64 | 163.44 | 4.16 | 3.49 | 39.29 | 46.83 |
| 80 | 205.19 | 5.14 | 4.26 | 39.92 | 48.17 |
| 96 | 251.75 | 5.94 | 4.89 | 42.38 | 51.48 |
| 112 | 294.96 | 6.96 | 5.73 | 42.38 | 51.48 |

The above table gives the time required to classify one of the test documents. When comparing the three implementations of Naïve Bayes', it is guaranteed that all of them classify the test document to the same class generating the same values for the posterior class probabilities. As is clear from Table 5.4, as the number of classes increase so does the gap between the time required by serial Naïve Bayes on CPU and the parallel Naïve Bayes' implementations on GPU. Figure 5.2 gives a plot of the execution time required by the three implementations.

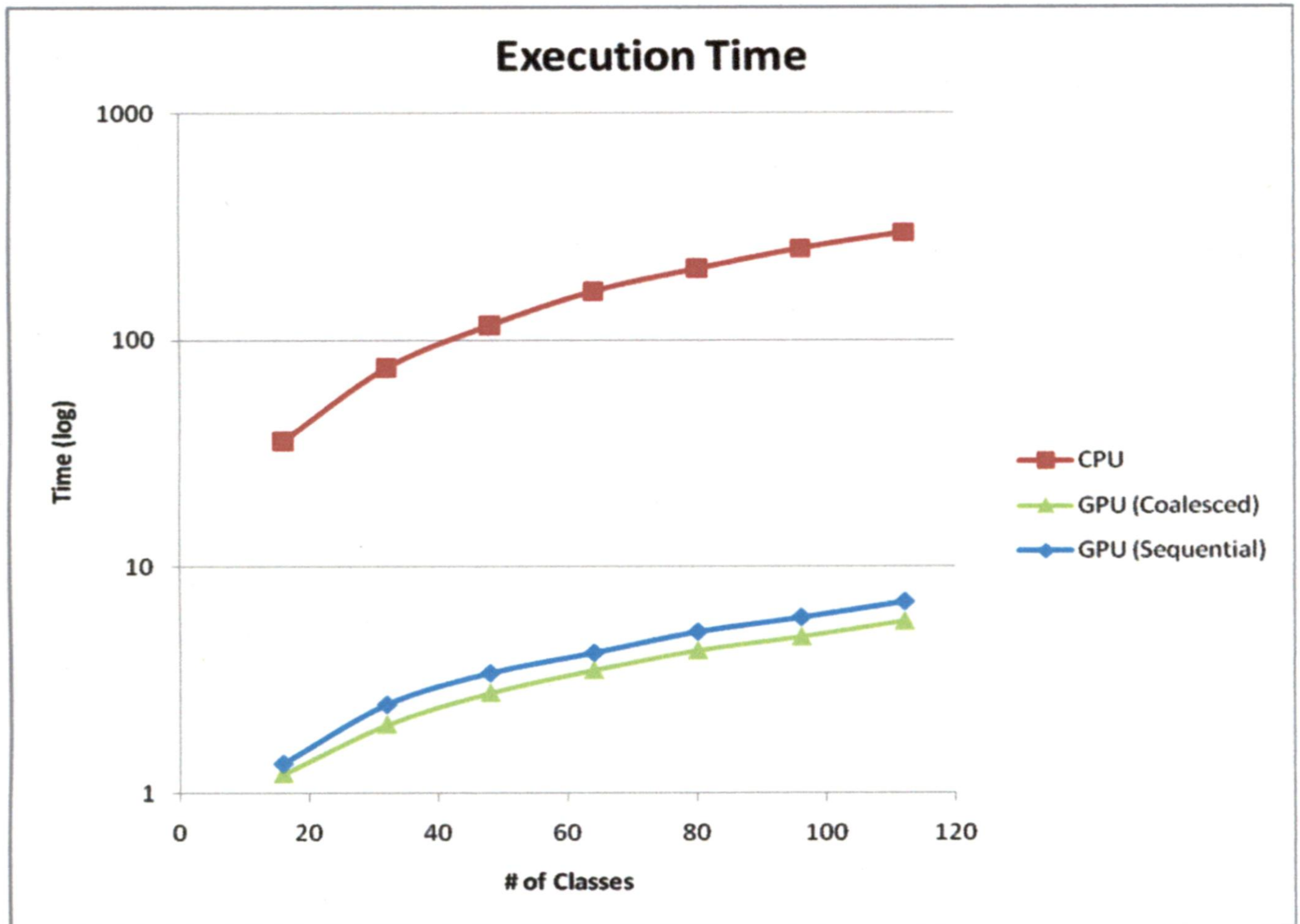



**Figure 5.2 Execution Time of Naïve Bayes using CPU, GPU (sequential memory accesses) and GPU (coalesced memory accesses) by varying number of classes**

Figure 5.3 gives a plot of the speed up obtained by the parallel implementations of Naïve Bayes' text classification on GPU over the implementation on CPU. It is clear that the implementations of Naïve Bayes' text classification using GPU are much faster than the

implementations using CPU. It is also observed that the parallel Naive Bayes' using coalesced memory access is much faster than the one using sequential memory accesses.

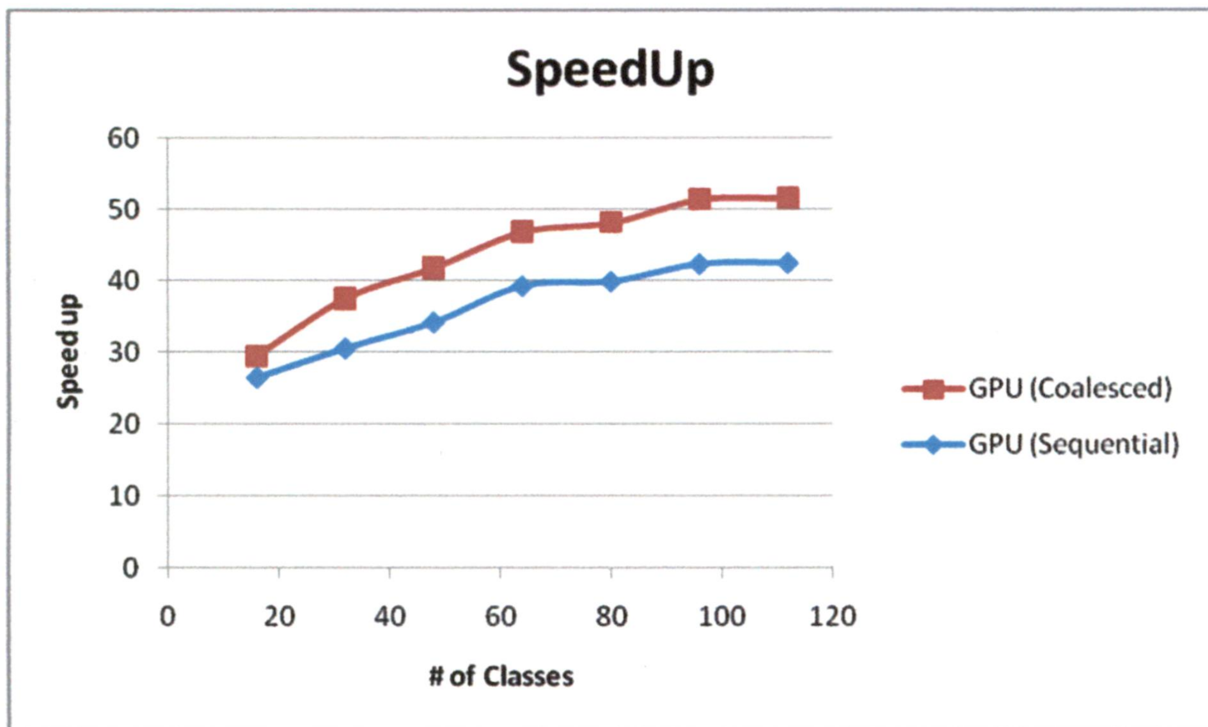



**Figure 5.3 Speed up obtained by using GPU as compared to CPU**

The figure shows that speedup by using the GPU reaches around 42%. When the memory accesses were coalesced, the GPU was used in a more concurrent manner and the speed increases reaching as high as approximately 52%.

### 5.2.3 Comparison between the two parallel Naïve Bayes' implementations

The time required by each implementation of parallel Naïve Bayes' text classification using the GPU is a sum of the following three components:

1. Time required to transfer the class-conditional word probabilities, the class prior probabilities and the feature vector to the GPU memory from the CPU memory
2. Time required for execution of kernel
3. Time required for transferring the posterior class probabilities from the GPU memory to the CPU memory.

Table 5.5 gives the breakup of the time required by the parallel Naïve Bayes' text classification algorithm using GPU. The time required in 1 and 3 have been merged and are together called as the data transfer time. The data transfer time for the two parallel implementations of Naïve Bayes' remains the same. The two implementations differ in their kernel execution times. Hence, to compare the two implementations only their kernel executions times have been considered. Figure 5.4 gives a plot of the execution time of the two implementations of parallel Naïve Bayes'. As is clear from Figure 5.4, the coalesced memory accesses greatly reduce the kernel execution time and therefore increase the speed of the parallel Naïve Bayes' classification for unstructured text data.

**Table 5.5 Comparison of Execution Time of the two parallel implementations of Naïve Bayes'**

| Number of classes | Data transfer time (ms) DT | Sequential Memory Access | | Coalesced Memory Access | | Speed Up (SKT/CKT) % |
|---|---|---|---|---|---|---|
| | | Kernel Execution time (ms) SKT | Total Time (ms) (DT + SKT) | Kernel Execution time (ms) CKT | Total Time (ms) (DT + CKT) | |
| 16 | 0.9 | 0.46 | 1.36 | 0.32 | 1.22 | 143.75 |
| 32 | 1.55 | 0.92 | 2.47 | 0.46 | 2.01 | 200 |
| 48 | 2.13 | 1.26 | 3.39 | 0.64 | 2.77 | 196.88 |
| 64 | 2.58 | 1.58 | 4.16 | 0.91 | 3.49 | 173.63 |
| 80 | 3.19 | 1.95 | 5.14 | 1.07 | 4.26 | 182.24 |
| 96 | 3.63 | 2.31 | 5.94 | 1.26 | 4.89 | 183.33 |
| 112 | 4.24 | 2.72 | 6.96 | 1.49 | 5.73 | 182.55 |

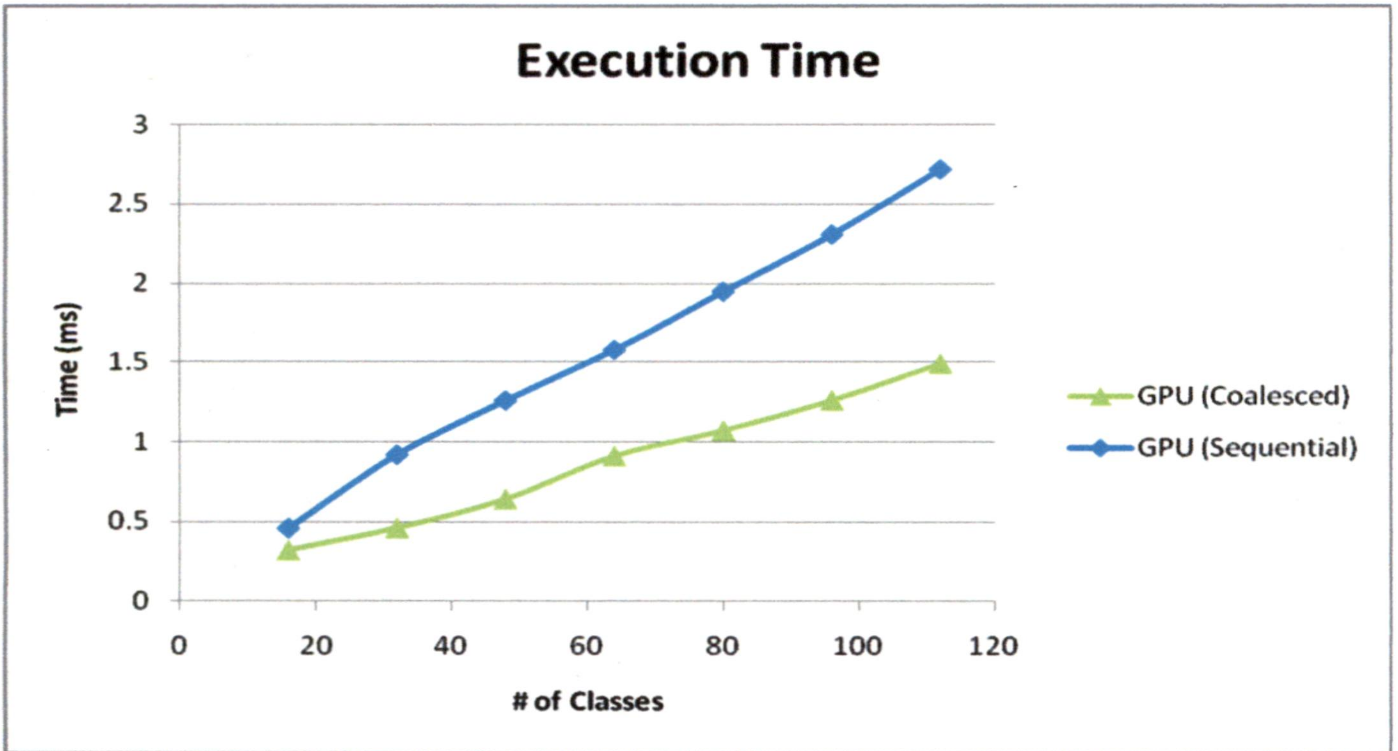



**Figure 5.4 Kernel Execution Time of Naïve Bayes using GPU (sequential memory accesses) and GPU (coalesced memory accesses) by varying number of classes**

# Chapter 6
# Conclusions and Future Work

## 6.1 Conclusions

In this dissertation, a parallel implementation of Naïve Bayes' text classification using CUDA has been proposed. The algorithm was modified to make use of WordNet synsets and POS tagging. The following conclusions can be drawn from this dissertation work:

- The classification step of Naïve Bayes' text classification was parallelized using CUDA. The proposed approach used the global memory, and the data accessed by each thread was sequential. Experimental results showed that the parallel implementation over GPU was up to 40 times faster than the implementation over CPU.
- The use of memory coalescing to reduce the number of memory accesses resulted in a decrease in the time required for classification. Experimental results show that this implementation was up to 52 times faster than the implementation over CPU and up to 1.75 times faster than the previous approach.
- The use of WordNet and POS tagging to capture the semantic information resulted in an increase in the classification accuracy of the proposed parallel Naïve Bayes' text classification algorithm. However, the improvements obtained were not very significant and ranged between 3-4% over the traditional method.

## 6.2 Scope for Future Work

There are some points where the proposed system's functionality can be extended and improved. The possible improvements in future are listed below:

- The parallelisation of the training module of Naïve Bayes' classifier can be explored for applications using the incremental model of learning. In such systems the classifier is trained frequently and hence parallelisation is a good option to reduce the amount of time spent in training.
- The parallelisation of the pre-processing module can also be explored. Most of the time in the proposed system is spent in pre-processing the text documents.
- Currently, the system has been designed such that only after a document has been processed completely, will another document be accepted for classification. But pipelining can be used such that when the GPU is busy calculating the posterior class probabilities, the CPU can load another text document and pre-process it.
- The use of WordNet and POS tagging has been done a rather simplistic way to avoid unnecessary complexities in the system. But they can be integrated into the proposed system in a more sophisticated way which could lead to greater improvements in classification accuracy.

# REFERENCES


[1] Tom Mitchell, Machine Learning, Tata McGraw Hill, 97

[2] T. Joachims. "Learning to Classify Text Using Support Vector Machines- Methods, Theory and Algorithms", in The Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, 2001.

[3] C. Silva, B. Ribeiro, "The importance of stop word removal on recall values in text categorization," *Neural Networks, 2003. Proceedings of the International Joint Conference on* , vol.3, no., pp. 1661- 1666 vol.3, 20-24 July 2003.

[4] (February 10). *Stemming and lemmatization.* Available: http://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

[5] G. A. Miller, "WordNet: A Lexical Database for English," *Communications of the ACM,* vol. 38, pp. 39-41, 1995.

[6] (April 28). *Stanford Log-linear Part-Of-Speech Tagger.* Available: http://nlp.stanford.edu/software/tagger.shtml

[7] J. Han and M. Kamber, Data Mining: Concepts and Techniques, 2nd ed., Elseveir, 2006, pp. 290-350.

[8] A. McCallum and K. Nigam, "A comparison of event models for Naive Bayes text classification," in *AAAI-98 Workshop on Learning for Text Categorization,* 1998, pp. 41-48.

[9] Karl-Michael Schneider, "A Comparison of event Models for Naive Bayes Anti-spam E-mail Filtering" 2003.

[10] NVIDIA Corporation. NVIDIA CUDA programming guide, Version 2.0, July 2008.

[11] NVIDIA Corporation. NVIDIA CUDA programming guide, Version 3.1, May 2010.

[12] Y. Zhang, F. Mueller, X. Cui, T. Potok, "GPU-Accelerated Text Mining," in Workshop on Exploiting Parallelism using GPUs and other Hardware-Assisted Methods, 2009.

[13] S. Liang, C. Wang, Y. Liu and L. Jian , "CUKNN: A parallel implementation of K-nearest neighbor on CUDA-enabled GPU," *YC-ICT '09: IEEE Youth Conference on Information, Computing and Telecommunication, 2009.* , vol., no., pp.415-418, 20-21 Sept. 2009

[14] C. Kruengkrai and C. Jaruskulchai, "A parallel learning algorithm for text classification," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, Edmonton, Alberta, Canada, pp. 201-206, 2002.

[15] Wang Ding; Songnian Yu; Qianfeng Wang; Jiaqi Yu; Qiang Guo; , "A Novel Naive Bayesian Text Classifier," *Information Processing (ISIP), 2008 International Symposiums on*, vol., no., pp.78-82, 23-25 May 2008.

[16] V. G. Roncero, M. C. A. Costa, and N. F. F. Ebecken, "Text classification on a grid environment," in *Proceedings of the 9th international conference on High performance computing for computational science*, Berkeley, CA, 2011, pp. 251-262.

[17] M. Rodríguez, J. Gómez-Hidalgo, and B. Agudo. Using WordNet to complement training information in text categorization. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing*, 1997

[18] S. Scott and S. Matwin, "Text Classification Using WordNet Hypernyms," in *Proceedings of the conference on Use of WordNet in Natural Language Processing Systems*, pp. 38–44, 1998.

[19] Lee S. Jensen, Tony Martinez. 2000. "Improving Text Classification by Using Conceptual and Contextual Features."

[20] Lin Lv; Yu-Shu Liu; , "Research of English text classification methods based on semantic meaning," *Information and Communications Technology, 2005. Enabling Technologies for the New Knowledge Society: ITI 3rd International Conference on* , vol., no., pp.689-700, 5-6 Dec. 2005

[21] Jian Yang, Mei Sun and Wenjun Zhou, "Study on Massive Text Classification Mining Grid System," *2nd International Symposium on Information Engineering and Electronic Commerce (IEEC), 2010*, vol., no., pp.1-6, 23-25 July 2010.