

**DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES**

**DICTIONARY-BASED EFFECTIVE AND
EFFICIENT TURKISH LEMMATIZER**

by
Mert CİVRİZ

September, 2011
İZMİR

DICTIONARY-BASED EFFECTIVE AND EFFICIENT TURKISH LEMMATIZER

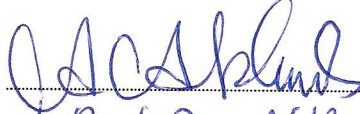
**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University In
Partial Fulfillment of the Requirements for the Degree of Master of Science in
Computer Engineering, Computer Engineering Program**

**by
Mert CİVRİZ**

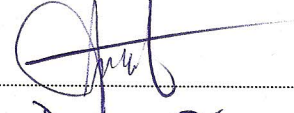
**September, 2011
İZMİR**

M.Sc THESIS EXAMINATION RESULT FORM

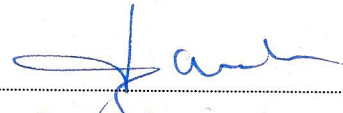
We have read the thesis entitled “**DICTIONARY-BASED EFFECTIVE AND EFFICIENT TURKISH LEMMATIZER**” completed by **MERT CİVRİZ** under supervision of **ASSIST. PROF. DR. ADİL ALPKOÇAK** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.


Asst. Prof. Dr. Adil ALPKOÇAK

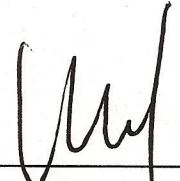
Supervisor


Asst. Prof. Dr. Derya BİRANT

(Jury Member)


Asst. Prof. Dr. Randa KANTEF

(Jury Member)


Prof. Dr. Mustafa SABUNCU
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to thank to my thesis advisor Assist. Prof. Dr. Adil Alpkoçak for his help, suggestions, patience and systematic guidance throughout the all formation phases of this thesis.

Furthermore, I would like to thank to Aslan Türk for his motivations, advices and help through my graduate school years.

And my special thanks go to my family; the most valuable asset of my life; for all their support, patience and happiness they gave me throughout my life.

Mert Civriz

DICTIONARY-BASED EFFECTIVE AND EFFICIENT TURKISH LEMMATIZER

ABSTRACT

In this thesis, we present a new Turkish lemmatizer that runs on the GPU and investigate its accuracy and performance. Turkish is an agglutinative language, with a rich morphological structure, contains homographic and inflectional word forms which are lowering the accuracy of stemmers. Thus, in Turkish information retrieval systems, the ability to lemmatize Turkish words efficiently and effectively is important. Our study aims at developing a fast dictionary based lemmatizing approach for indexing and searching documents in Turkish.

Recent introduction of CUDA (Compute Unified Device Architecture) libraries for high performance computing on graphic processing units (GPUs) by NVIDIA has increased the trend to use GPUs as general purpose performance environment (GPGPU). Today researchers started to exploit GPU's high computational capability through CUDA in many applicative contexts requiring intensive use of computational resources such as molecular dynamics, fluid dynamics, cryptology, computer vision, astrophysics and genetics.(e.g. Manavski and Valle, 2008) CUDA can be used also in the information retrieval because of its massively workload. Our program, achieves a speedup of as much as 90 times on a recent GPU (NVIDIA GeForce GT240M) over the equivalent CPU-bound version, ultimately with the use of parallelized execution of lemmatization algorithm using a data structure inspired from "Radix Trie". Here, we present evaluation results of our string lemmatizing kernels for use in CUDA, which executes parallelized lemmatizing for a test set of query strings. We compared our lemmatization algorithm running on GPU with the serial CPU bound version, and explored issues associated with efficient use of GPU resources with eight different algorithms.

Keywords: Information Retrieval, Turkish Information Retrieval, Lemmatizer, CUDA, GPGPU, Parallel Programming

SÖZLÜK TABANLI ETKİN VE VERİMLİ TÜRKÇE GÖVDELEYİCİ

ÖZ

Bu çalışmada, GPU üzerinde çalışan bir Türkçe gövdeleyici algoritması geliştirdik ve daha sonra bu algoritmanın performansını ve verimliliğini araştırdık. Türkçe sondan eklemeli ve zengin morfolojik yapıya sahip bir dil olarak eşesli ve yapısal değişkinliğe uğrayabilen kelimeleri içerdiği için sözlük kullanmadan sadece kurallar tanımlanarak gövdeleme yapılması zahmetli ve verimsiz olacaktır. Bu yüzden Türkçe bilgi getirim sistemlerinde, Türkçe kelimelerin etkin ve verimli bir şekilde sözlük tabanlı gövdelenmesi önemlidir. Bu çalışmamız Türkçe dökümanların indekslenmesi ve aranması amacıyla sözlük tabanlı hızlı bir gövdeleyici geliştirmeyi amaçlıyor.

Yüksek performanslı programlama amacıyla Nvidia tarafından tanıtılmış, grafik programlama üniteleri üzerinde çalışan ve hala geliştirilmekte olan CUDA kütüphanesi grafik programlama ünitelerinin, grafik programlamanın dışında genel amaçlı performans ortamı olarak kullanılması eğilimini arttırdı. Bugünlerde, araştırmacılar hesaplama kaynaklarının yoğun olarak kullanılmasını gerektiren moleküler dinamikler, akışkan dinamikleri, kriptoloji, görüntü işleme, astrofizik ve genetik gibi bir çok alanda CUDA ile grafik programlama ünitelerinin yüksek hesaplama kabiliyetinden yararlanmaya başladı.(Manavski ve Valle, 2008 gibi) CUDA bilgi getirim işlemlerinin doğasında olan büyük iş yükleri için de kullanılabilir. Bizim programımız GPU üzerinde (NVIDIA GeForce GT240M) “Radix Trie” veri yapısı mantığıyla geliştirilen gövdeleyici algoritmasının paralel çalışırılması ile CPU üzerinde çalışan seri versiyonuna göre, 90 kata kadar performans artışı sağladı. Bu tezde, kelime gövdeleyici algoritmalarımızın test kelime seti üzerinde çalıştırarak elde ettiğimiz sonuçları gösteriyoruz. GPU üzerinde çalışan gövdeleyici algoritmamızı CPU üzerinde çalışan versiyonuyla karşılaştırdık ve GPU kaynaklarını nasıl daha verimli kullanılabileceğimizi sekiz farklı algoritmayla araştırdık.

Anahtar Sözcükler: Bilgi Erişimi, Türkçe Bilgi Erişimi, Gövdeleyici, CUDA, GPGPU, Paralel Programlama

CONTENTS

	Page
M.Sc. THESIS EXAMINATION RESULT FORM.....	ii
ACKNOWLEDGMENTS	iii
ABSTRACT	iv
ÖZ	v
CHAPTER ONE - INTRODUCTION	1
1.1 Introduction	1
CHAPTER TWO - LEMMATIZATION	4
2.1 Lemma.....	4
2.1.1 Difference between stem and lemma.....	4
2.2 Lemmatization.....	5
2.3 Turkish Lemmatization	5
2.3.1 Morphological Structure of Turkish Words	5
2.3.2 Structure Of Dictionary	8
2.3.3 Data Structure Selection	10
2.3.4 Lemmatization Algorithm	19
CHAPTER THREE - GPU AND GPGPU	26
3.1 GPU	26
3.1.1 GPU Architecture	28
3.2 GPGPU	32
CHAPTER FOUR - CUDA	33

4.1 CUDA Overview	33
4.2 CUDA Programming Model	34
4.2.1 CUDA Kernels.....	34
4.2.2 Thread Model.....	36
4.2.3 Memory Model	39
4.2.3.1 Global memory	41
4.2.3.2 Local Memory.....	41
4.2.3.3 Shared memory	42
4.2.3.4 Registers.....	42
4.2.3.5 Constant Memory.....	42
4.2.3.6 Texture Memory	43
4.3 CUDA Optimization Strategy	43
4.3.1 Instruction Throughput	43
4.3.1.1 Arithmetic Instructions	43
4.3.1.2 Control Flow Instructions	44
4.3.1.3 Memory Instructions.....	44
4.3.2 Memory Bandwidth.....	45
4.3.2.1 Data Transfers between Host and Device.....	46
4.3.2.2 Global Memory Accesses	46
4.3.2.3 Local Memory.....	47
4.3.2.4 Constant Memory.....	47
4.3.2.5 Texture Memory	47
4.3.2.6 Shared Memory.....	47
4.3.2.7 Registers.....	48
4.3.3 Occupancy	48
CHAPTER FIVE - LEMMATIZATION ON GPU	50
5.1 Lemmatization Algorithm on CUDA.....	50
5.1.1 Redesigning Structure.....	50
5.1.2 Occupancy	56

CHAPTER SIX - EVALUATION	59
6.1 Test Data and Measurement Method.....	59
6.1.1 Test Data.....	59
6.1.2 Measurement.....	59
6.2 Evaluation of Lemmatizer Accuracy.....	60
6.2.1 Precision at N documents	66
6.2.2 Precision – Recall Averages	67
6.2.3 Map, Gmap and Rprec	68
6.2.4 Bpref	69
6.3 Evaluation of Lemmatizer Performance.....	70
6.3.1 Parameters.....	70
6.3.2 Methods	71
6.3.3 Results	71
 CHAPTER SEVEN - CONCLUSION AND FUTURE WORK	 76
 REFERENCES	 79
 APPENDICES	 83

CHAPTER ONE

INTRODUCTION

1.1 Introduction

With dramatic expand of Internet technology, computer users generating new data for their requirements on the web so online data that the information retrieval based on is increasing rapidly. Along with these growth; information retrieval deals on large-scale documents that are created for different purposes in many different languages by numerous users. Information retrieval (IR) works for classifying, indexing and searching on this huge amount of data. As the necessity of this, various approaches are applied to address this issue for indexing, retrieval and ranking, some of them are kept secret due to commercial benefits. Stemming and lemmatizing methods are only some of these approaches. In addition to these approaches, more specific, language dependent methods are required to improve results. For this purpose the major points of a language that differ from others must be determined. In particular, for Turkish, we come up with the differences of Turkish Alphabet and the grammar structure for suffixes. Word structures can grow to an unmanageable size because Turkish morphology is very complex and more over there are many exceptional cases in Turkish. From the point of the differences of Turkish Morphology, a lemmatizer is a need for accurate IR programs.

Lemmatizers play a significant role in information retrieval (Frakes & Baeza-Yates, 1992). The ability to lemmatize words efficiently and effectively is thus important. Lemmatization is used in the IR for listing all the morphological variants of a word. Usually, this is done by looking up a list of related words in a dictionary. This kind of lemmatization is computationally simpler, since almost all the work is done off-line in compiling the dictionary of morphological variants. Lemmatization is another normalization technique where for each inflected word form in a document or request, its basic form, the lemma, is identified. The benefits of lemmatization are the same as in stemming. In addition, when basic word forms are used, the searcher may match an exact search key to an exact index key. Such accuracy is not possible with truncated, ambiguous stems.

Within the field of internet technology and growing online data there is an increasing demand for faster ways to solve a variety of information retrieval and natural language processing problems, for some of which Compute Unified Device Architecture (CUDA) might be the right answer due to its scalable programming model. CUDA is still relatively new and evolving rapidly and with its each new release the computational abilities of the devices grow and it becomes easier to exploit their computational power.

Graphics Programming Units (GPUs) differ from general-purpose microprocessors in their design for utilizing the Single Instruction Multiple Data (SIMD) paradigm. Due to the inherent parallelism of graphic programming, GPUs adopted multicore architectures long before regular processors evolved to such a design. As a result, today GPUs consist of many small computation cores that support a higher number of floating-point operations per second. Originally designed to accelerate computer graphics applications through massive on-chip parallelism, GPUs have evolved into powerful platforms for more general purposes of compute-intensive tasks, called as GPGPU (General Purpose Graphic Programming Unit). Given their extremely high workload, information retrieval provides a very interesting potential application domain for GPUs. NVIDIA's launch of the CUDA with its simple but effective programming model has resulted in the adoption of GPUs by a diversity of domains. The NVIDIA CUDA programming model takes its power from this simplicity, much in contrast to the previous approaches of GPGPU environments. With CUDA, programmers no longer have to master graphics specific knowledge, before being able to efficiently program GPUs. It has been demonstrated that CUDA can significantly speed-up many computationally intensive applications from domains such as scientific computation, physics, molecular dynamics simulation, genetics, imaging and the finance sector.

In this thesis, we introduce a Turkish lemmatizer works on GPGPU through NVIDIA's CUDA. Building an efficient IR lemmatizer for GPUs is a non-trivial task due to the branching and diverging nature of lemmatizing algorithm and hardware constraints provided by the GPU. We outline and discuss a general architecture of our lemmatizer and later we present our studies on GPU-based version of lemmatizer

with different performance optimization techniques. Finally, we compare CPU-bound and GPU-bound versions of our algorithm and make a performance analysis.

This thesis is divided into seven chapters. The next chapter, chapter two, reviews lemmatization process briefly and in addition to that explains our data structure selection phases and implementation of lemmatizing algorithms in detail.

Chapter three introduces the GPU and GPGPU architecture and illustrates how they work. It is important to know development environment to use it efficiently.

Chapter four identifies CUDA, its programming model and abstractions, and also required works to achieve higher speed up rate.

Chapter five gives information about our studies of parallelization and redesigning of algorithm in order to achieve an efficient lemmatizer on GPU.

Chapter six is about experiments and results on a selected dataset in two sub-chapters. In first part, we looked at accuracy of our lemmatizer and later we measured performance of it.

Finally last chapter, chapter seven, discusses results, concludes and gives a look to possible future research studies.

CHAPTER TWO

LEMMATIZATION

2.1 Lemma

In linguistics, a lemma (from the Greek noun “*lēm̄ma*”, “headword”) is the “dictionary form” or “canonical form” of a set of words. More specifically, a lemma is the canonical form of a *lexeme* where *lexeme* refers to the set of all the forms that have the same meaning, and *lemma* refers to the particular form that is chosen as base form to represent the lexeme. In information retrieval, this unit is usually also the *citation form* or headword by which it is indexed. Lemmas have special significance in highly inflected and agglutinative language such as Turkish.

In a dictionary-based lemmatizer, a lemma can be seen as the headword of a dictionary entry. Where, a dictionary entry consists of two parts:

- the lemma,
- the information of the lemma.

2.1.1 Difference between stem and lemma

In computational linguistics, a stem is the part of the word that never changes even when morphologically inflected, whilst a lemma is the base form of the word. For example, with a “fixed prefix truncate by 4 characters” stemmer extracts stem as “boyn” from the word “boynu” where the lemma is “boyun”. During searching, the retrieval system using this stemmer most probably return documents related to “boynuz” (horn) since they will share the same stem “boyn”. In linguistic analysis, the stem is defined more generally as the analyzed base form from which all inflected forms can be formed.

2.2 Lemmatization

Lemmatization is the process of determining the lemma for a given word, so different inflected forms of a word can be analyzed as a single item. Lemmatization is the process which creates the set of lemmas of a lexical database. It is conceived as starting from text-words found in a corpus and leading to lemmas heading dictionary entries.

Lemmatization is related to stemming but unlike stemming, which operates only on a single word at a time, lemmatization operates on the full text and therefore can discriminate between words that have different meanings depending on part of speech. On the other hand, stemmer operates on a single word without knowledge of the context that chops off the ends of words, and often includes the removal of derivational affixes. Therefore stemmers cannot discriminate between words, which have different meanings depending on part of speech. However, stemmers are typically easier to implement and run faster, and the reduced accuracy may not matter for some applications. The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

In our case, *dictionary-based lemmatizer*, lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is corresponding to the lemma.

2.3 Turkish Lemmatization

2.3.1 Morphological Structure of Turkish Words

Stemming and lemmatizing is an essential task for indexing and information retrieval purposes in agglutinative languages. Turkish is also an agglutinative language, which has a rich morphological structure. Words are usually composed of

a stem and of at least two or three affixes appended to it. And this is why it is usually harder analyze a Turkish text.

In linguistics, a *morpheme* is the smallest meaningful component of a word and morphology is analysis and description of the structure of morphemes. Morphology is also interested in how morphemes can be combined to form words. For example if we analyze the word “tezim” (“my thesis” in English) we see that it has two units. One of them is main meaning of word. In this example “tez” is the main meaning of the word. This morpheme is called stem; and the remaining morpheme which is “im” in this example is called as affix.

In Turkish, there are two kinds of processes to combine morphemes to form words: *inflection* and *derivation*. Word structures are formed by affixations of derivational and inflectional suffixes to stems.

Inflectional process is adding grammatical affixes to word stem. It doesn't change the class of word. Unlike English nouns, which have only two kinds of inflection (plural and possessive); there are more kinds of inflectional affixes in Turkish.

For example the word “arabalar” (“cars” in English) can be broken down into morphemes as follows:

“araba” + “-lar”

where the +’s indicate morpheme boundaries. Here “araba” (“car” in English) and “arabalar” are both nouns.

Derivational process is simply an affix addition to a word stem which will change the meaning and in some cases the class of the stem. For example when we break the word “gözlük” (“eye glasses” in English) into morphemes:

“göz” + “-lük”

the affix “-lik” is a derivational morpheme. It changes the meaning of the word while it doesn’t change the class of stem. The words “göz” (“eye” in English) and “gözlük” are both nouns.

Some derivational affixes can change both words meaning and class. For example when we look at morphemes of the word “öğretmen” (“teacher” in English):

“öğret” + “-men”

the affix “-men” is a derivational morpheme in the word “öğret” (“to teach” in English). It changes both the meaning of the word and class of the stem. The word “öğretmen” is a noun while the word “öğret” is a verb.

There are two main classes for Turkish roots. These classes are nominal and verbal. Morphemes added to a root word can convert the word from a nominal to a verbal structure (vice versa) or can create adverbial constructs. Under some circumstances vowels in the roots and morphemes may be deleted depending on the affix (vowel deletion / haplology). Similarly consonants in the roots words or in the affixed morphemes may get through some modifications and may sometimes be deleted. These two rules are presented below:

- **Last consonant alteration**

If last letter of a word or suffix is a stop consonant (süreksiz sert sessiz), and a suffix that starts with a vowel is appended to that word, last letter changes (voicing). Changes are **p-b, ç-c, k-ğ, t-d, g-ğ**.

Some last consonant alteration examples are : *kitap*→*kitab-a*, *pabuç*→*pabuc-u*, *cocuk*→*cocuğ-a*, *hasat*→*hasad-ı*, *garp*→*garbı*

And with some suffixes: *elma-cık*→*elma-cığ-ı*, *yap-acak*→*yap-acağ-ım*

When a word ends with “nk”, then “k” changes to “g” instead of “ğ”:
cenk→*ceng-e*, *çelenk*→*çeleng-i*

For some loan words, g-ğ change occurs: *psikolog*→*psikoloğ-a*

- **Vowel deletion (vowel ellipsis or haplology)**

Last vowel before the last consonant drops in some words when a suffix starting with a vowel is appended: *ağız*→*ağz-a*, *burun*→*burn-um*, *zehir*→*zehr-e*, *nakit*→*nakd-e*, *lütuf*→*lütf-un*

Also some verbs obeys this rule: *kavur*→*kavr-ul*

2.3.2 Structure Of Dictionary

The dictionary we have used for our work is “*Büyük Türkçe Sözlük*” (Grand Turkish Dictionary), the one that is published by TDK (Turkish Language Association) and it is open to public via internet (<http://tdkterim.gov.tr/bts/>). This dictionary lists the senses along with their definitions and example sentences that are provided for some senses.

“*Büyük Türkçe Sözlük*” consists different kinds of dictionaries like science terms, art terms, sports terms, place names, regional dialects, etc. A typical entry from this dictionary for the word “*tez*” (has two meanings : 1.fast 2.thesis) is given below in Figure 2.1:

(I) 1. Çabuk olan, süratli. 2. Süratli bir biçimde.
<i>Güncel Türkçe Sözlük</i>
(II) 1. Sav. 2. Üniversitelerde öğrencilerin veya öğretim üyelerinin hazırlayıp bazen bir sınav kurulu önünde savundukları bilimsel eser: “ <i>Tezini mitolojiden hazırlayan gözlüklü bir delikanlı.</i> ” - H. Taner.
<i>Güncel Türkçe Sözlük</i>

Figure 2.1 Dictionary entry for query word “tez”

The entry in the dictionary has the following information:

(II) . (sense number) / 2. (subsense) / Üniversitelerde öğrencilerin veya öğretim üyelerinin hazırlayıp bazen bir sınav kurulu önünde savundukları bilimsel eser (definition) / “Tezini mitolojiden hazırlayan gözlüklü bir delikanlı.” (example sentence) / - H. Taner. (citation) / Güncel Türkçe Sözlük (dictionary type)

As is seen, in Turkish, a word commonly has more than one meaning. In order to work efficiently we parsed and analyzed all the entries on “Büyük Türkçe Sözlük” then inserted them into a database table. Later the dictionary in the database is used for word (lemma) and sense enumeration of it for standardization. More specifically, we parsed and inserted the information on previous entry of dictionary (on Figure 2.1) into database as follows:

Table 2.1 Representation of dictionary on database table

ID	OrderNo	Word	Meaning	Dictionary Type
342864	311713	tez	Çabuk olan, süratli.	Güncel Türkçe Sözlük
342865	311713	tez	Süratli bir biçimde.	Güncel Türkçe Sözlük
342867	311713	tez	Sav.	Güncel Türkçe Sözlük
342868	311713	tez	Üniversitelerde öğrencilerin veya öğretim üyelerinin hazırlayıp bazen bir sınav kurulu önünde savundukları bilimsel eser	Güncel Türkçe Sözlük

Here it can be seen that “tez” has four meanings (on Table 2.1) in database while the entry is divided into two meanings in “Büyük Türkçe Sözlük” (Figure 2.1). While constructing the database we parsed all meanings into separate records with having different “ID” but having same “OrderNo” on identical lemma. Thus, we can access to and use lemma’s all different meanings with only its “OrderNo” field and can

select appropriate meaning for use of word sense disambiguation algorithms via its unique “ID”.

2.3.3 Data Structure Selection

When we were thinking for the best possible data structure that is suitable for our needs; our design goals were:

- The data structure should support prefix searching.
- The data structure should store thousands of entries with a low space complexity (must be suitable with the architecture constraints of GPU discussed on Chapter Three).
- The data structure should be able to store prefixes with variable lengths in each node.
- The data structure should be fast (because we seek through thousands of words in dictionary).
- Look-up method of data structure should not be data dependent and recursive (must be suitable with constraints of CUDA detailed on Section 4.1).
- The data structure should be suitable with Turkish language’s rich agglutinative structure.
- The data structure should be suitable with our finite state machine implementation discussed on Section 2.6.

After a little survey we decided on *trie* structure which is suitable for our requirements because the way tries are space efficient since nodes are shared between keys with common prefixes, facilitates longest-prefix matching, and also can be seen as a deterministic finite automaton with regard to its manner of work pattern.

Tries (name comes from reTRIEval trees) are tree-based structures where each node represents a part of the key. A trie is an ordered tree structure that is used to

store a collection of the keys, which are usually strings. All the descendants of a node have a common prefix of the string associated with that node.

Table 2.2 A sample list of Turkish words

Words	
Doğmak	Dokunak
Doğum	Dokunaç
Doku	Dokunma
Dokuma	Dokunmak
Dokumacı	Dokunmatik
Dokumak	Dokunulmaz

For instance, a trie would store the list of Turkish words presented in Table 2.2 as follows:

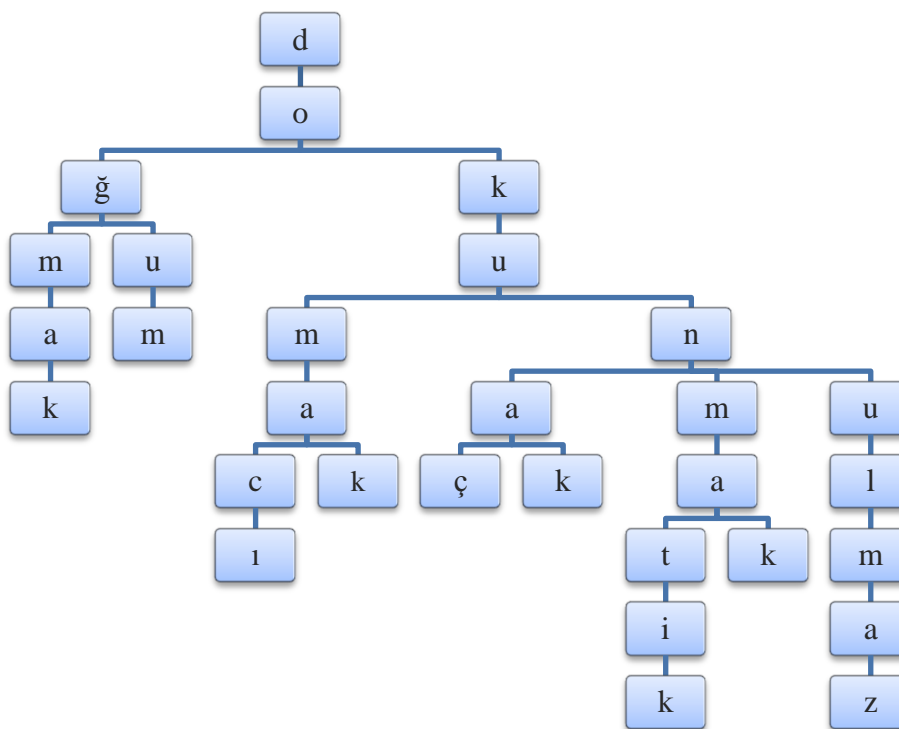


Figure 2.2 Visual representation of the words' settlement on trie in Table 2.2.

There are several variants of the trie data structure, one of the most efficient being the PATRICIA (Practical Algorithm To Retrieve Information Coded In Alphanumeric) trie, which is also known as “Radix” trie (Morrison, 1968).

The main characteristic of the radix trie is the way it eliminates unnecessary nodes by grouping the sequences of keys whenever possible. Each node with only one child is merged with its child. The result is that every internal node has at least two children. Unlike in regular tries, edges can be labeled with sequences of characters as well as single characters. This makes them much more efficient for sets of strings that share long prefixes.

Using a Radix trie, the words in Table 2.2 would be inserted as Figure 2.3 below:

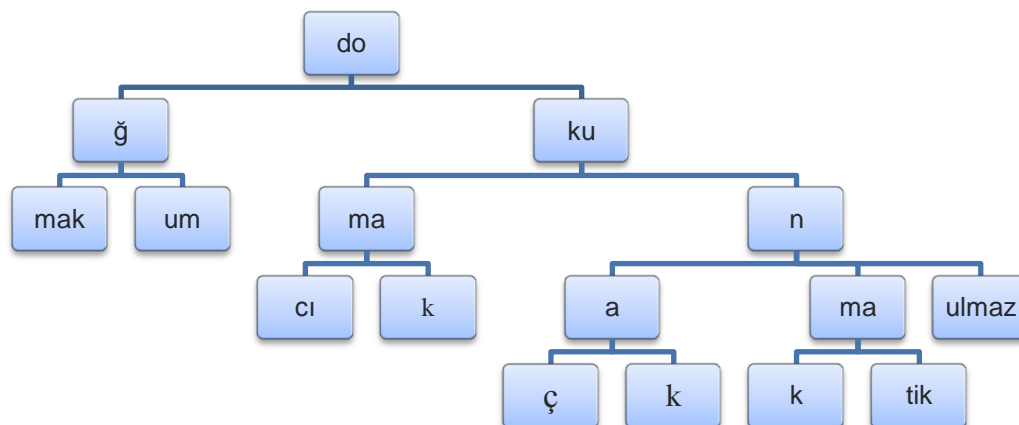


Figure 2.3 Radix Trie allocation for given set of words

Radix tries can be constructed time affiliated to the length of the corpus, and provide exact matching of a query in time proportional to the length of the query, independent of the size of the corpus.

Basically, radix trie is a compact data structure that can give you the longest prefix of an entry key in $O(N)$ steps (in the worst case), with N the length of the longest prefix.

For instance, the look-up method used with radix trie, taking the following Turkish word “dokunmatik” as argument retrieves the object highlighted in the Figure 2.4 below:

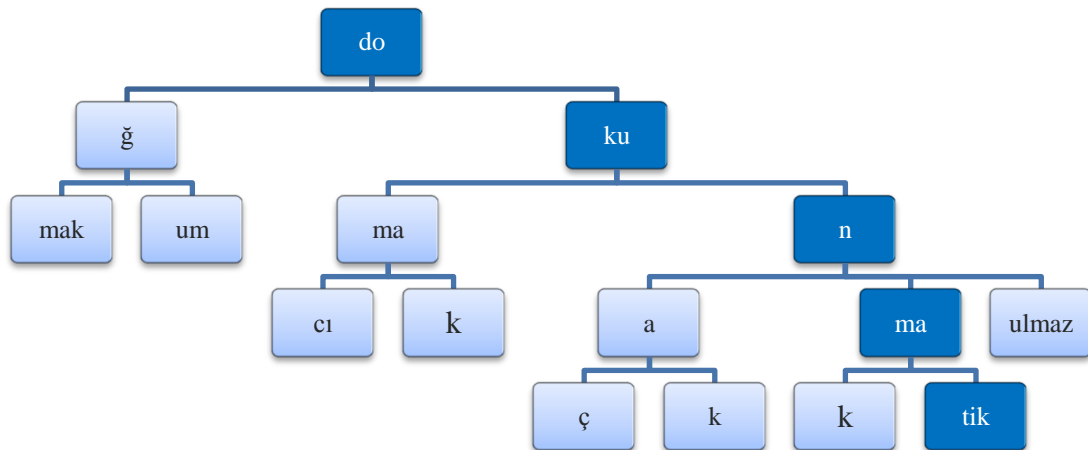


Figure 2.4 Look-up of {dokunmatik} in PATRICIA Trie

We first designed our structure as a digital radix trie that holds keys on external nodes and binary representation of characters on trie but then; to adopt the rules haplology (vowel deletion) and consonant alteration we implemented the trie to work on characters instead of binary numbers.

In order to prepare our dictionary for selected structure, we stored the parsed and analyzed lemmas (dictionary entries) and their extracted features from their information in the database into a XML like formatted file, which would be helpful for designing our structure. Because XML style annotation increases readability and allows manual addition to corpus with simple text editors or code snippets.

We have divided the information on the database records into two XML files. One to hold meanings of lemmas named as “Dictionary Data XML” and the other one named as “Trie Data XML” to hold headwords of the lemmas, thinking the fact that our lemmatizer doesn’t need meanings of words for its purpose. The structure of the “Dictionary Data XML” can be seen below in Figure 2.5.

```

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<root>
  .
  .
  .
  <RECORD ID="342864" orderno="311713" meaning="tez"
  anlam="Çabuk olan, süratli." type="Güncel Türkçe Sözlük"/>
  <RECORD ID="342865" orderno="311713" meaning="tez"
  meaning="Süratli bir biçimde." type="Güncel Türkçe Sözlük"/>
  <RECORD ID="342867" orderno="311713" word="tez"
  meaning="Sav." type="Güncel Türkçe Sözlük"/>
  <RECORD ID="342868" orderno="311713"
  word="tez" meaning="Üniversitelerde öğrencilerin veya
  öğretim üyelerinin hazırlayıp bazen bir sınav kurulu
  önünde savundukları bilimsel eser" type="Güncel
  Türkçe Sözlük"/>
  .
  .
  .
</root>

```

Figure 2.5 Representation of word “tez” in Dictionary Data XML

More elaborately, in “Dictionary Data XML”, ”**word**” stands for lemma itself (“*word*” field in database) and the “**ID**” field in the XML corresponds to the lemma’s “*ID*” on database table and likewise “**orderno**” corresponds to “*OrderNo*” in the database and finally “**type**” represents dictionary type (“*DictionaryType*” field in database). The “**ID**” field differs on each record but “**orderno**” field stays same on identical lemma (**word**) which is conceptually parallel with the database table formation.

The structure of the XML file which provides lemmas (can be seen in Figure 2.6) for lemmatizer contains prefix information and basic level morphological analysis of the words. If a word has a corresponding meaning in the dictionary or is a common prefix of more than one word in the dictionary; it is stored as a separate node. Here if a node has a corresponding meaning in “Dictionary Data XML” it’s “**orderno**”

property stored as “**Data**” property of node in “Trie Data XML”. Similarly, if a node is available for a consonant alteration; the alteration affix had been saved into “**ConsAlterKey**” property. “**MasterData**” and “**MasterKey**” were added in order to hold the verb meaning and verb version respectively for the cases that a lemma has more than one meaning. We simply unify these two versions into one lemma but separate meanings. For example, assuming the analyzing / parsing procedure meets with word “oymak”, the procedure will save the meaning of “oy” (“vote” in English and is a noun) into “**Data**” property, the meaning of “oy (mak)” (“to drill” in English and is verb) into “**MasterData**” property and the suffix “mak” into “**MasterKey**”. Finally “**VowelDeletion**” was added to hold the information that if a node is available for haplology or in other words, can be skippable in order to search its sub nodes. The consonant alteration keys and vowel deletion datas are not added manually. These properties added automatically via an algorithm by analyzing all of the lemma’s morphemes on “Trie Data XML” file’s constructing time. The resulting corpus is 14.31MB and has 137372 nodes. The structure of XML can be seen below:


```

<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<root>
  .
  .
  <t ConsAlterKey="" MasterKey="" VowelDeletion="0" Data=""
  MasterData="">
    .
    .
    <e ConsAlterKey="" MasterKey="mek" VowelDeletion="0"
    Data="306029" MasterData="308662">
      .
      .
      <z ConsAlterKey=""
      MasterKey="mak" VowelDeletion="0"
      Data="311713" MasterData="" />
      .
      .
    </e>
    .
    .
  </t>
  .
  .
</root>

```

Figure 2.6 Representation of word "tez" in "Trie Data XML"

After we formed our XML files, we defined our trie nodes with regard to XML formation. Each property of a XML node has a corresponding property on our trie node definition which are presented below:

- **Key:** This property holds actual key of node. (This property of node corresponds to "name" property of XML node)
- **ConsAlterKey:** This property holds the consonant alteration key of node's "key". This node will be null if key is not suitable for consonant alteration but will store the replacement key on other case. For example if key is "k" this

property will be “ğ” or “g” with regard to parent node. If parent node’s key ends with “n” then “ConsAlterKey” will be “g” otherwise “ğ”. (This property of node corresponds to “ConsAlterKey” property on XML node)

- **MasterKey:** This property doesn’t actually necessary for lemmatizing process but we need it when we use our lemmatizer on word sense disambiguation or query / document expansion (finding an appropriate synonym of word) purposes. Can be ”mak” or “mek” depending on prefix on parent node. (This property of node corresponds to “MasterKey” property on XML node)
- **Data:** This property holds dictionary order of the word. Like “MasterKey” this is only required when we need to get lemma’s meaning from dictionary and work on it. We use this property to decide whether the node’s key corresponds to a lemma when added to its prefixes. (This property of node corresponds to “Data” property on XML node)
- **MasterData:** Considering the fact that in Turkish a word can be used both as a verb and a noun this property holds verb meaning of some words having more than one meaning. For example: “oymak” has two meanings. “tribe/clan” (noun) and “to drill” (verb) so Data holds noun meaning and “MasterData” holds verb meaning. (This property of node corresponds to “MasterData” property on XML node)
- **VowelDeletion:** This property holds a boolean variable stating the node’s key is suitable for haplology. (This property of node corresponds to “VowelDeletion” property on XML node)
- **Children:** This property holds a pointer of node’s children.
- **ChildCount:** This property holds count of children of node.

So regard to this structure, the words on Table 2.2 settles to trie as follows:

Table 2.3 Representation of the words in Table 2.2 on our structure.

Node No	Key	ConsAlter Key	Master Key	Vowel Deletion	Data	Master Data	Children
0	Do	-	-	-	-	-	1,3
1	Do- ğ	-	mak	-	-	97914	2
2	Do-ğ- um	-	-	1	98331	-	-
3	Do- ku	-	mak	-	98598	98661	4,6
4	Do-ku- ma	-	-	-	98651	-	5
5	Do-ku-ma- ci	-	-	-	98654	-	-
6	Do-ku- n	-	mak	-	-	98722	7,10
7	Do-ku-n- a	-	-	-	-	-	8,9
8	Do-ku-n-a- k	ğ	-	-	98675	-	-
9	Do-ku-n-a- ç	c	-	-	98669	-	-
10	Do-ku-n- ma	-	-	-	98710	-	-

And a visual presentation of trie for an explicit view is shown below:

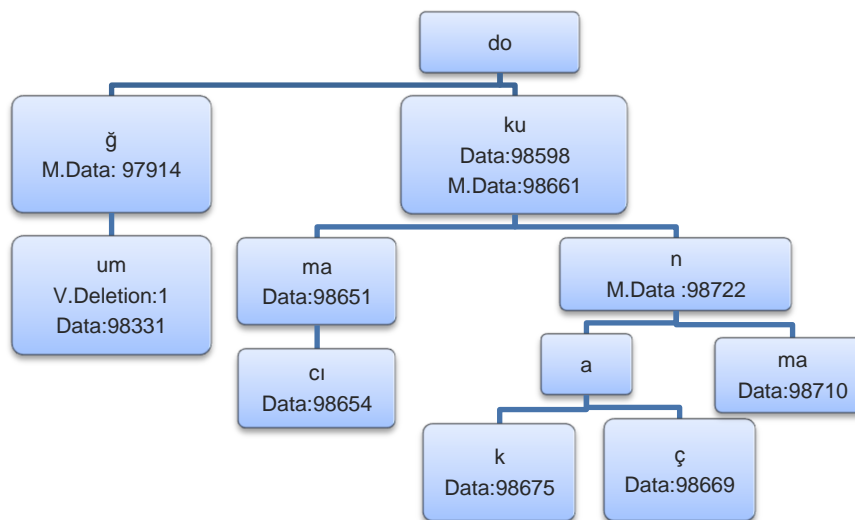


Figure 2.7 Our trie allocation of dictionary words shown in Table 2.3.

2.3.4 Lemmatization Algorithm

In Turkish, the suffixes are affixed to the stem according to definite ordering rules. The agglutinative and rule-based nature of word formations in Turkish allows modeling of the morphological structure of language in Finite State Machines (FSMs). In Figure 2.8 there is a finite state machine expressing the ordering rules of these suffixes based on our algorithm with a list of Turkish words in Table 2.4. The double circles on nodes represent the accept states of the FSM. A character on an arc indicates which suffix causes a state transition. And “any” on an arc represents the rest of the characters that is not indicated by any arc from current state. If there are multiple characters on an arc, all of the suffixes defined by those characters can cause that state transition. While traversing the FSM by consuming suffixes in each transition, reaching to an accepting state means that a possible stem is reached.

Table 2.4 A sample list of Turkish words

Words	
Doğmak	Dokumak
Doğum	Dokunak
Doku	Dokunaç
Dokuma	Dokunma
Dokumacı	Dokunmak

The finite machine in brief:

- accepts the string x if it ends up in an accepting state, and
- rejects x if it does not end up in an accepting state.

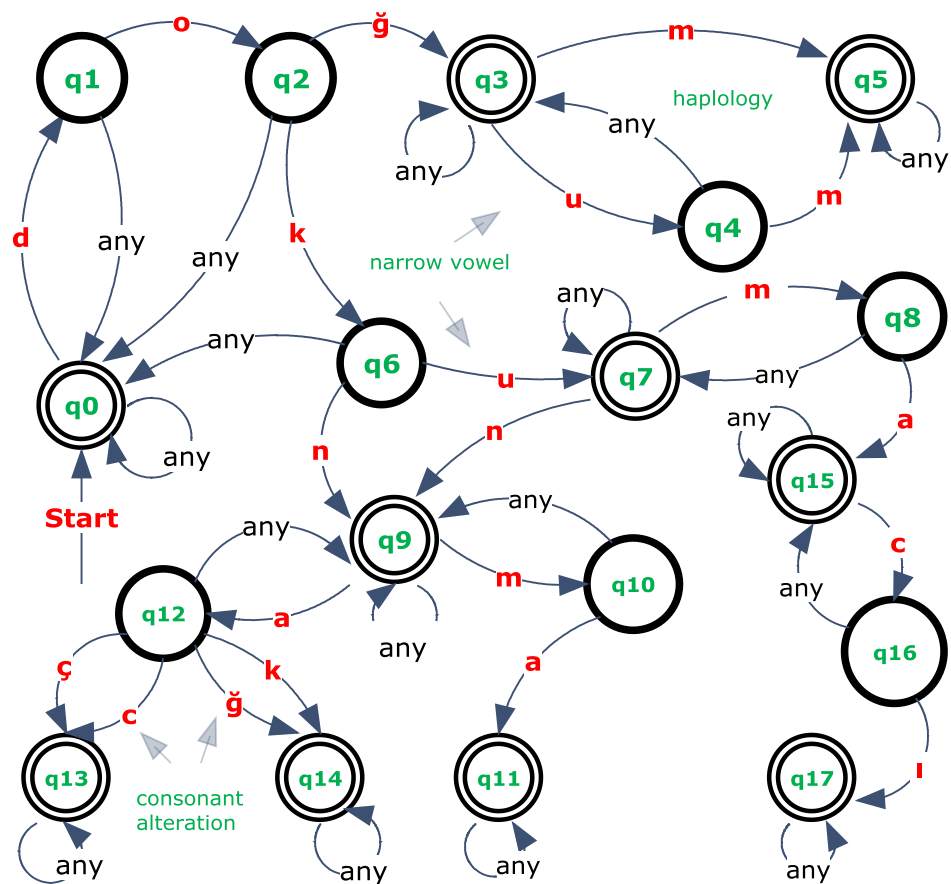


Figure 2.8 FSM representation of our lemmatizing algorithm for the words on Table 2.4.

Thus, for example if we give word “dokusu” as an input, FSM in Figure 2.8 starts with **q0**, then reads the word, character by character, changing state after each character read. When the FSM is in state **q0** and reads character “d”, it enters state **q1**. Then follows a route of **q1** → (*o*) → **q2** → (*k*) → **q6** → (*u*) → **q7**. After that it reads “s” and doesn’t change state since there is no state bound to “s”. Same happens for “u”. And after FSM consumes all characters; it accepts “doku” as lemma since it is an accepting state. Transition table of FSM in Figure 2.8 is shown on Table 2.5 below:

Table 2.5 State transition table of FSM in Figure 2.8

	d	o	ğ	k	u	m	a	c	ç	ı	n	Word
q0	<u>q1</u>	q0	q0	q0	q0	q0	q0	q0	q0	q0	q0	-
q1	q0	<u>q2</u>	q0	q0	q0	q0	q0	q0	q0	q0	q0	d
q2	q0	q0	<u>q3</u>	<u>q6</u>	q0	q0	q0	q0	q0	q0	q0	do
q3	q3	q3	q3	q3	<u>q4</u>	<u>q5</u>	q3	q3	q3	q3	q3	doğ
q4	q3	q3	q3	q3	q3	<u>q5</u>	q3	q3	q3	q3	q3	doğu
q5	q5	q5	q5	q5	q5	q5	q5	q5	q5	q5	q5	doğum
q6	q0	q0	q0	q0	<u>q7</u>	q0	q0	q0	q0	q0	<u>q9</u>	dok
q7	q7	q7	q7	q7	q7	<u>q8</u>	q7	q7	q7	q7	<u>q9</u>	doku
q8	q7	q7	q7	q7	q7	q7	<u>q15</u>	q7	q7	q7	q7	dokum
q9	q9	q9	q9	q9	q9	<u>q10</u>	<u>q12</u>	q9	q9	q9	q9	dokun
q10	q9	q9	q9	q9	q9	q9	<u>q11</u>	q9	q9	q9	q9	dokunm
q11	q11	q11	q11	q11	q11	q11	q11	q11	q11	q11	q11	dokunma
q12	q9	q9	<u>q14</u>	<u>q14</u>	q9	q9	q9	<u>q13</u>	<u>q13</u>	q9	q9	dokuna
q13	q13	q13	q13	q13	q13	q13	q13	q13	q13	q13	q13	dokunak
q14	q14	q14	q14	q14	q14	q14	q14	q14	q14	q14	q14	dokunaç
q15	q15	q15	q15	q15	q15	q15	q15	<u>q16</u>	q15	q15	q15	dokuma
q16	q15	q15	q15	q15	q15	q15	q15	q15	q15	<u>q17</u>	q15	dokumac
q17	q17	q17	q17	q17	q17	q17	q17	q17	q17	q17	q17	dokumacı

While we are taking the advantage of our dictionary based algorithm we did also consider some rules for more effective and accurate lemmatization. In Turkish, when a suffix is used, a letter may change into another one or may be discarded. For example, the change of “p” to “b” in example of “kitap” (“book” in English) and “kitaba” is an example of letter transformation (consonant alteration). And “burun” (“nose” in English) to “burnum” illustrates the second case since the letter *u* drops (vowel deletion). Our algorithm can handle both situations with some exceptions on the latter. Because a match is more important than transformation in our algorithm; we simply ignore the transformation when we find a match in current node’s children. Thus, the exceptions occur when there is a node key equals to transformation character. For example “kayıt” evolves into “kayda” with “-a” suffix, and in the dictionary there are lemmas like “kaydırmak” and “kaydetmek” which

consists the “*d*” transformed letter after their “*kay*” morpheme. So when procedure looks up for “kayda” in trie it encounters with “*d*” after “*kay*” (to slide) lemma. From this point, the procedure doesn’t look for a transformation and continues to its path on trie from “*d*” node, since there is a valid match. And it returns “*kay*” as lemma because there is no child node with “*a*” key after “*d*” node (there is no lemma as “kayda” in dictionary).

In summary, when user wants to lemmatize a word with our lemmatizer, lemmatization procedure starts searching characters of word from left to right and seeks them through in trie nodes. If a node key matches with current character or character sequence, then procedure checks whether the node has its “Data” or “MasterData” (has a meaning in dictionary) properties are occupied which determine the accepting states of our implementation. This process continues until the query has no more suffixes left to search; and at the end, latest lemma (accepting state) is returned as an output. Here is a pseudo code for simplified CPU-based version of our algorithm (Figure 2.9).

```

PROCEDURE LemmatizeWord(Trie,token,lemma)

    CurrentNode = Root of Trie;
    Buffer= array of 21 characters (longest Turkish word's length)
    MatchIndex= -1;
    MatchLength=0;
    HaplologyIndex=-1;

    WHILE CurrentNode NOT NULL DO

        IF CurrentNode HAS NOT any children
            THEN RETURN;
        ENDIF

        MatchIndex = -1;
        MatchLength = 0;
        HaplologyIndex = -1;

        FOR position = 0 TO ChildCount of CurrentNode DO

            CurrentChild = Node at position of CurrentNode's Children
            CurrentKey = Key of CurrentChild;
            CurrentConsKey = ConsonantAlterKey of CurrentChild;

```

We look that if current node's key or consonant alteration key, and token has a common prefix by a simple string compare algorithm

```

    CommonPrefixLength = GetCommonPrefix(CurrentKey,
    CurrentConsKey,token);

```

If we have a match then we break loop and proceed to second part of algorithm

```

    IF CommonPrefixLength > MatchLength
        THEN
            MatchLength = CommonPrefixLength;
            MatchIndex = position;
            BREAK LOOP;
        ENDIF

```

If there is no match we look if current node is suitable for haplology through its preprocessed Haplology property but we dont break loop because a match is more important than a haplology and succeeding nodes may contain a common prefix

```

    IF CurrentChild has narrow vowel
        THEN
            IF HaplologyIndex < 0
                THEN
                    HaplologyIndex = position;
                ENDIF
            ENDIF

```

(a)

If we don't have a match (MatchIndex equals to its initial value), we look if there is a haplology. If HaplologyIndex bigger than its initial value we move our pointer to current node's child node at haplology index and concatenate its key to the buffer. Otherwise it means we have reached the latest lemma, so we just assign buffer vrb. to lemma vrb. and return.

```

IF MatchIndex == -1
THEN
  IF HaplologyIndex > -1
  THEN
    CurrentNode = Node at HaplologyIndex of CurrentNode's Children
    CurrentKey = Key of CurrentNode;
    Buffer = Concatenate CurrentKey to Buffer;
  ENDIF
ELSE
  THEN
    IF Lemma IS NULL
    THEN
      Lemma = Buffer;
    ENDIF
  RETURN;
ENDIF

```

If we have a match, the procedure continues from here. And firstly we delete common prefix from token.

```

TokenLength = length of token;
token = substring of token from MatchLength to TokenLength;

```

Later we move the pointer to the child node at MatchIndex of current node's children and concatenate current node's key to the buffer.

```

CurrentNode = Node at MatchIndex of CurrentNode's Children
CurrentKey = Key of CurrentNode;
Buffer = Concatenate CurrentKey to Buffer;
CurrentData = Data of CurrentNode;
CurrentMasterData = MasterData of CurrentNode;

```

And finally we look if current node has a corresponding meaning in the dictionary. If current node's Data or MasterData properties are not NULL it means we have an accept state and a possible lemma. So we assign buffer to lemma variable.

```

IF CurrentData IS NOT NULL OR CurrentMasterData IS NOT NULL
THEN
  Lemma = Buffer;
ENDIF

```

If all the characters in word is consumed then quit and return with latest lemma.

```

IF Character length of Token == 0
THEN RETURN;
ENDIF

```

```

ENDWHILE
ENDPROCEDURE

```

(b)

Figure 2.9 (a) is the first part and (b) is the second part of the pseudo code of the CPU-bound version of lemmatizing algorithm

For example when we want to lemmatize token “*tezim*”; the lemmatizing procedure detailed with pseudocode on Figure 2.9 will follow the steps shown below on Table 2.6:

Table 2.6 Steps taken while lemmatizing word "tezim"

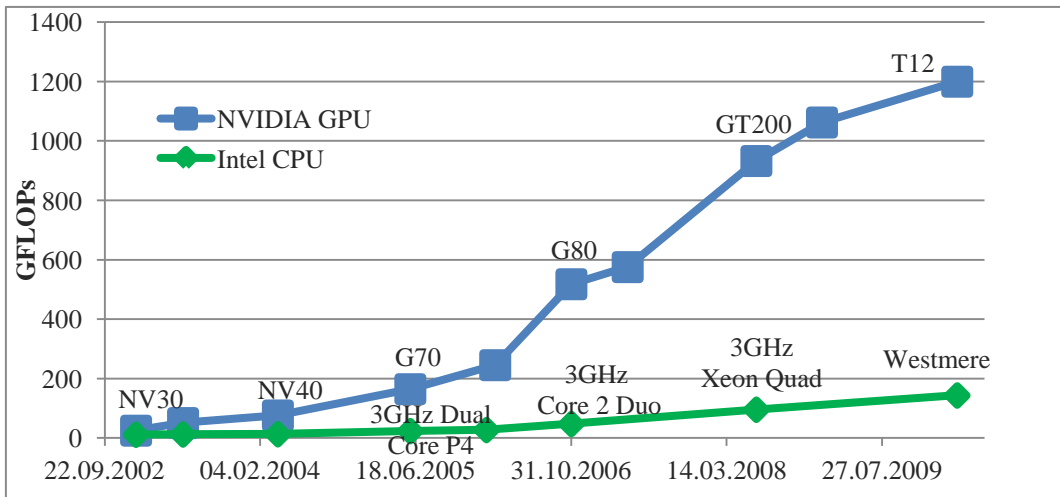
Step	Buffer	Current Key	Lemma	Token	Current Data	Current Data	Master
1	-	root	-	tezim	-	-	-
2	-	t	-	tezim	-	-	-
3	t	t	-	ezim	-	-	-
4	t	e	-	ezim	306029	30862	-
5	te	e	te	zim	-	-	-
6	te	z	te	zim	311713	-	-
7	tez	z	tez	im	-	-	-
8	tez	No match	tez	im	-	-	-

The procedure starts to search “*tezim*” on trie (**Step 1**). The first match happens at the node which has “*t*” key (**Step 2**). Following this match the key (“*t*”) is concatenated to buffer and is deleted from token which leaves token equal to “*ezim*”. Then the procedure looks if the current node has its “Data” property occupied. Current node (“*t*”) has no data property so procedure continues to search “*ezim*” through the child nodes of it (**Step 3**). The next match comes up at node “*e*” (child of node with key “*t*”) (**Step 4**). Here node with key “*e*” has its “Data” property (“*te*” has a meaning in dictionary) not null. So, key “*e*” is concatenated to buffer and then the content of buffer is assigned to lemma. Later procedure starts to search the resulting token “*zim*” through child nodes (**Step 5**). When the procedure comes at node with key “*z*” a match happens; and because “*z*” has a valid “Data” property the steps done at node “*e*” are repeated for node “*z*” and these left token as “*im*” (**Step 6 and 7**). The token “*im*” has no match with the child nodes of “*z*”, so lemma and buffer doesn’t change and procedure returns the lemma as “*tez*” along with the dictionary meaning as “311713” which is the last accepting state (**Step 8**).

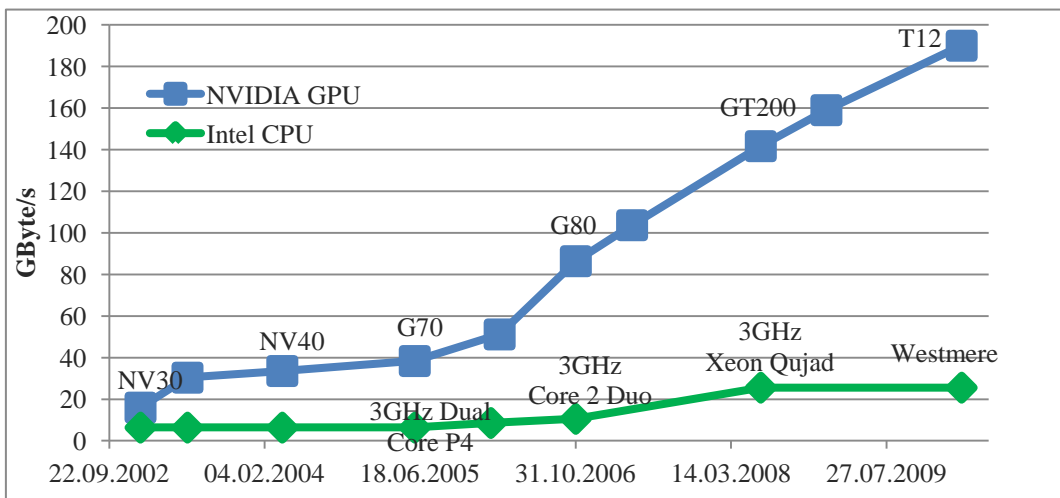
CHAPTER THREE GPU AND GPGPU

3.1 GPU

A graphics processing unit or GPU is a processor attached to a graphics card dedicated to calculating floating point operations. GPU has evolved into a highly parallel, multithreaded; many core processor with tremendous computational power and very high memory bandwidth, as illustrated by Figure 3.1.



(a)



(b)

Figure 3.1 Floating-point operations per second (a) and memory bandwidths of the CPU and GPU (b) (NVIDIA Corporation, November 2010).

The reason behind the divergence in floating-point capability (FLOPS) between the CPU and the GPU is that the CPU evolved to be good at any problem whether it is parallel or not and performs best when small pieces of data are processed in a complex, but sequential way. This lets the CPU to utilize the many transistors used for caching, branch prediction and instruction level parallelism. On the other hand the GPU is specialized for compute intensive, highly parallel workloads (massively data parallel problems) to work efficiently and therefore designed such that more transistors are devoted to data processing rather than data caching and flow control, as schematically illustrated by Figure 3.2.

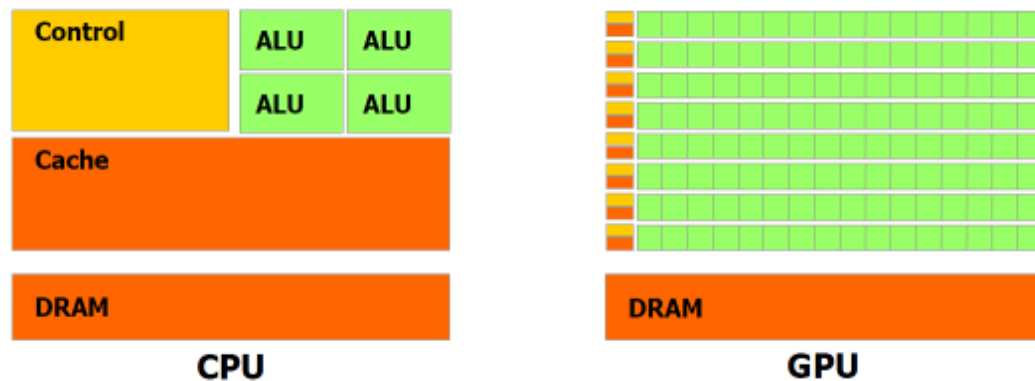


Figure 3.2 The GPU devotes more transistors to Data Processing (NVIDIA Corporation, November 2010)/

More specifically, the GPU is designed to address problems that can be expressed as data parallel computations, since the program works in SIMD fashion, with high arithmetic intensity. Also there is a lower requirement for sophisticated flow control and, the program is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches.

CPU's execution units can support a limited number of concurrent threads. Today servers with four quad-core processors can run only 16 threads concurrently (32 if the CPUs support Hyper Threading). On the other hand GPUs can support from 768 to more than 30000 active threads (NVIDIA Corporation, August 2010).

In addition that above, CPU threads are heavyweight entities. The operating system must swap threads' state (on and off) on CPU execution channels to provide multithreading (e.g. round - robin). Thus; context switching is slow and expensive. On the contrary threads running on GPUs are extremely lightweight. Because all active threads have their own separate memory registers, so no swapping of registers or state need occur between GPU threads.

Both the host system and the device have their own random access memory (RAM). On the host system, RAM is generally equally accessible to all code. On the device, RAM is divided virtually and physically into different types, each of which has a special purpose and fulfills different needs.

Another important difference between a CPU and a typical GPU is the memory bandwidth. Because of simpler memory models and no requirements from legacy operating systems, the GPU can support more than 180 GB/s of memory bandwidth, while the bandwidth of CPUs is around 20 GB/s (in Figure 3.1.b).

3.1.1 GPU Architecture

The GPU is a many core processor containing an array of streaming multiprocessors (SMs). A SM contains an array of streaming processors (SP), along with two more processors called special function units (SFUs). Each SFU has four floating point (FP) multiply units which are used for transcendental operations (e.g. sin, cosine) and interpolation. There's a MT issue unit that dispatches instructions to all of the SPs and SFUs in the group. In addition to the processor cores in a SM, there's a very small instruction cache, a read only data cache and a 16KB read/write shared memory (NVIDIA Corporation, November 2010). The units can be seen in Figure 3.3 below.

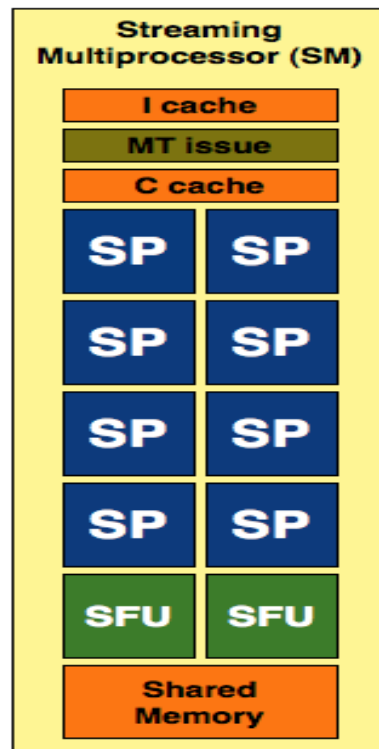


Figure 3.3 Streaming Multiprocessor
(Shimpi & Wilson, 2008)

A streaming processor (SP) is a fully pipelined, single-issue, in-order microprocessor, built with two arithmetic logic units (ALU) and a floating point unit (FPU) (Figure 3.4). But a SP doesn't have any cache, so it's not particularly great at anything other than computing tons of mathematical operations (Shimpi & Wilson, 2008).

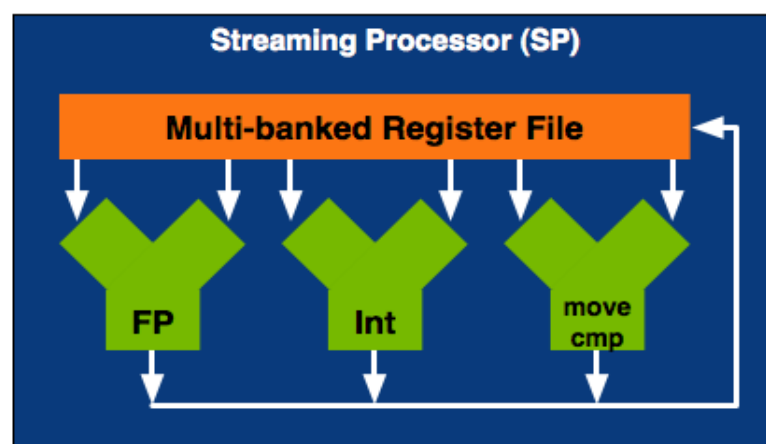


Figure 3.4 Streaming Processor (Shimpi & Wilson, 2008)

Each SM manages multithread allocating and scheduling as well as handling divergence through an instruction scheduling unit (MT issue). SM maps each thread to an SP for execution where each thread maintains its own register state. After this point threads have all the resources they need to run, threads can launch and execute basically for free. So all the SPs in a SM execute their threads in lock-step, according to the order of instructions issued by the scheduler. The SM creates and manages threads in bundles called as warps (NVIDIA Corporation, November 2010).

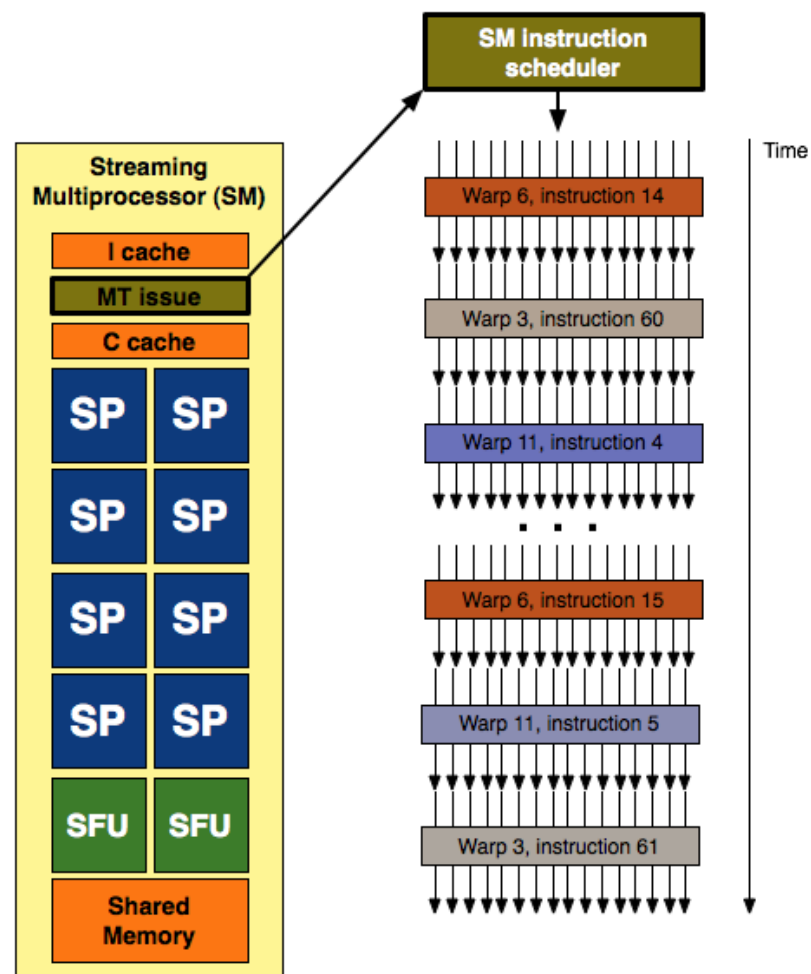


Figure 3.5 Scheduling of warps on SM (Shimpi & Wilson, 2008)

A warp is the smallest unit of scheduling within each SM. In SIMT fashion, threads are assembled into groups of 32 called “warps” which are simultaneously executed on different SPs at hardware level. Threads in warps share the control logic (i.e. the current instruction). Thus, every thread within a warp must be executing the

same instruction but different warps built from threads executing the same program can follow completely independent paths down the code. This means that branch granularity is 32 threads; every warp are allowed to can branch independently of all others (divergence), but if one or more threads within a warp branch in a different direction than the rest then every single thread in that warp must execute both code paths. Resolving divergence is also automatically handled by the hardware. The GPU achieves efficiency by splitting its work-load into multiple warps and multiplexing many warps onto the same SM (Figure 3.5). When a warp that is scheduled attempts to execute an instruction whose operands are not ready (e.g. an incomplete memory load), the SM switches context to another warp that is ready to execute, thereby hiding the latency of slow operations such as memory loads. Each SM can have 32 warps in work at the same time (NVIDIA Corporation, November 2010).

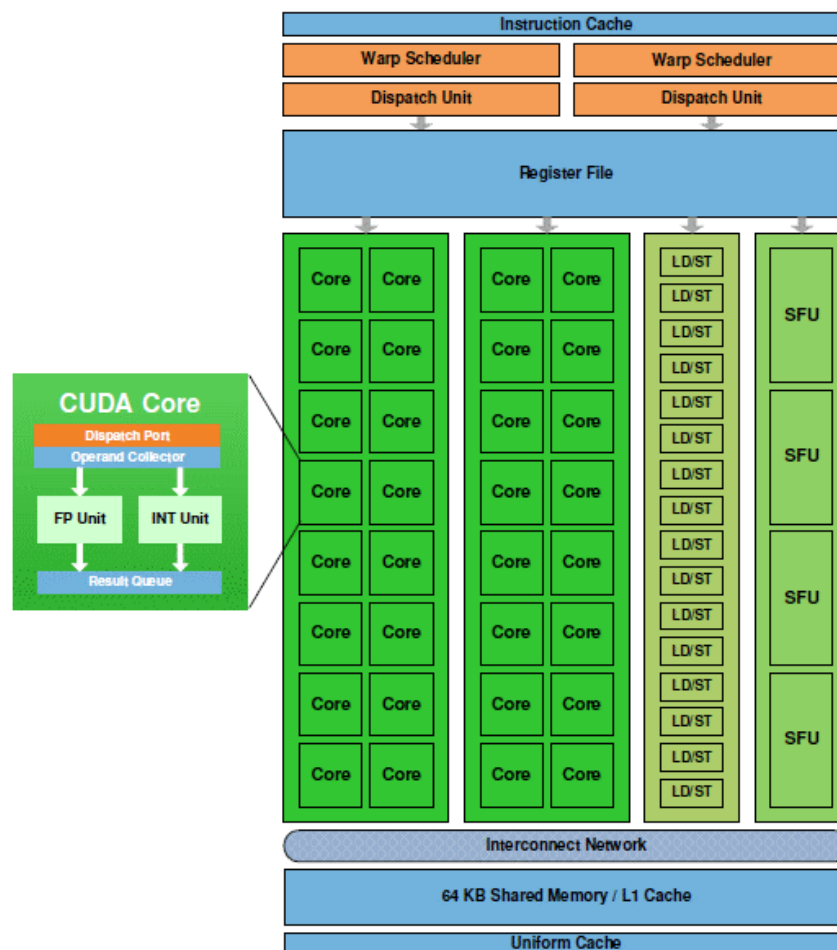


Figure 3.6 Nvidia Fermi GPUs' architecture (NVIDIA Corporation, 2009)

To sum it up, a CUDA compatible GPU architecture is shown above in Figure 3.6. In NVIDIA's CUDA compatible Fermi GPU architecture, a SM is made up of two SIMD 16-way units. Each SIMD 16-way has 16 SPs, thus a SM in Fermi has 32 SPs or 32 CUDA cores and 64KB shared memory (NVIDIA Corporation, 2009).

3.2 GPGPU

General purpose graphics processing units (GPGPU) offers new opportunities for the information retrieval community. GPUs are highly optimized towards the types of operations needed in graphics, but GPU vendors have recently started to allow researchers to exploit their computing power for other types of applications. Modern GPUs offer large numbers of computing cores (48 cores in NVIDIA GeForce GT240M, 512 Cores in NVIDIA Fermi) that can perform many operations in parallel, plus a very high memory bandwidth (memory throughput) that allows processing of large amounts of data (NVIDIA Corporation, November 2010). However, to be efficient, computations need to be carefully structured to conform to the programming model offered by the GPU, which is a data-parallel model reminiscent of the massively parallel SIMD (single instruction multiple data) fashion. Recently, GPU vendors have started to offer better support for general-purpose computation on GPUs. One major vendor of GPUs, NVIDIA, recently introduced the Compute Unified Device Architecture (CUDA), a new hardware and software architecture that simplifies GPU programming.

CHAPTER FOUR

CUDA

4.1 CUDA Overview

CUDA (Compute Unified Device Architecture) is a general-purpose hardware interface designed to let programmers exploit NVIDIA graphics hardware for general purposes instead of graphics programming. CUDA provides a programming model and well defined programming abstracts (e.g. memory model, thread model) that are consistent between all CUDA devices. The programming model describes how parallel code is written, launched and executed on a device via defining model a virtual model of GPU architecture allowing users a direct access to corresponding hardware. Thread model presents a thread hierarchy on how threads works and the memory model defines the different types of memories that are available to a CUDA program.

The functional paradigm of CUDA views the GPU as a coprocessor to the CPU. The GPUs supporting this language also facilitate scattered (arbitrary addresses) memory transactions in GPU which are essential for GPUs to operate as a general-purpose computational machine.

CUDA has several advantages (NVIDIA Corporation, November 2010) over traditional computation models on GPUs (GPGPU):

- Code can read/write from and to arbitrary addresses in memory (scattered transaction).
- A fast shared memory region that can be shared amongst threads which enables higher bandwidths.
- Faster read / write operations from and to the GPU
- Full support for integer and bitwise operations.

But those advantages come with some limitations (NVIDIA Corporation, November 2010) presented below:

- CUDA does not allow recursions and function pointers.
- Transferring the data between the CPU and the GPU is slow due to the bus bandwidth and latency.
- The SIMD execution model becomes a significant limitation for any divergent task (i.e. divergent branches in the code).
- CUDA is only available on NVIDIA GPU's.

4.2 CUDA Programming Model

The programming model most commonly used when programming a GPU is based on the stream programming model. In the stream programming model, input to and output from a computation comes in the form of streams. A stream is a collection of homogeneous data elements on which some operation, called a kernel, is to be performed, and the operation on one element is independent of the other elements in the stream.

In CUDA programming model there are three key abstractions which are a hierarchy of thread groups, shared memories, and barrier synchronization. These abstractions guide the programmer to partition the problem into sub problems that can be solved independently in parallel, and then into finer pieces that can be solved cooperatively in parallel. Each sub-problem can be scheduled to be solved on any of the available processor cores: A compiled CUDA program can therefore execute on any number of processor cores, and only the runtime system needs to know the physical processor count.

4.2.1 CUDA Kernels

In CUDA, GPU is modeled as a collection of streaming multiprocessors (SM) which work in Single Program Multiple Data (SPMD) fashion. With regard to this model, programmer writes a kernel and then the programming model generates lots

of threads that execute the same kernel, each working on a different set of data in parallel (NVIDIA Corporation, November 2010). A CUDA kernel is a function that is executed on a large set of data elements, shown in Figure 4.1

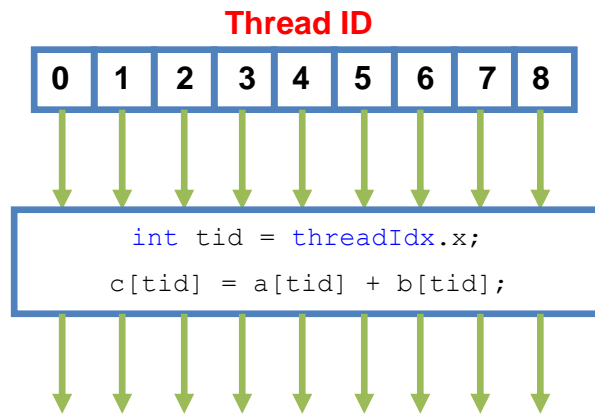


Figure 4.1 Kernel Execution

In this model, the programmer writes two separate kernels for a GPGPU application: code for the GPU kernel and the code for the CPU kernel. The CPU kernel must proceed through five general stages:

1. Allocate necessary input and output data space in GPU memory.
2. Transfer input data from host (CPU) memory to the GPU.
3. Call the GPU kernel wait until GPU kernel finishes its work. GPU kernel is executed parallel in each core.
4. Transfer the output data back to host memory from the GPU's memory.
5. Free allocated data space from GPU memory.

In brief, the GPU kernel is a sequence of instructions that directs each GPU thread to perform necessary operations on a unique data element in cause of the concurrent execution of all GPU threads in a SIMD (single-instruction, multiple-data) workflow.

These kernels are dynamically dispatched and executed in bundles of threads on SIMD multiprocessors. At any given clock cycle, each processor executes the identical kernel instruction on a thread bundle, but each thread operates on distinct data.

4.2.2 Thread Model

There are two important differences between GPU threads and CPU threads. First, there is no cost to create and destroy threads on the GPU. Additionally, GPU multiprocessors perform context switches between thread bundles (analogous to process switching between processes on a CPU) with zero latency. Both of these factors enable the GPU to provide its thread-level parallelism with very low overhead.

The CUDA programming model organizes threads into a three-level hierarchy as shown in Figure 4.2. At the highest level of the hierarchy is the grid. A grid is a two dimensional array of thread blocks, and thread blocks are in turn three dimensional arrays of threads.

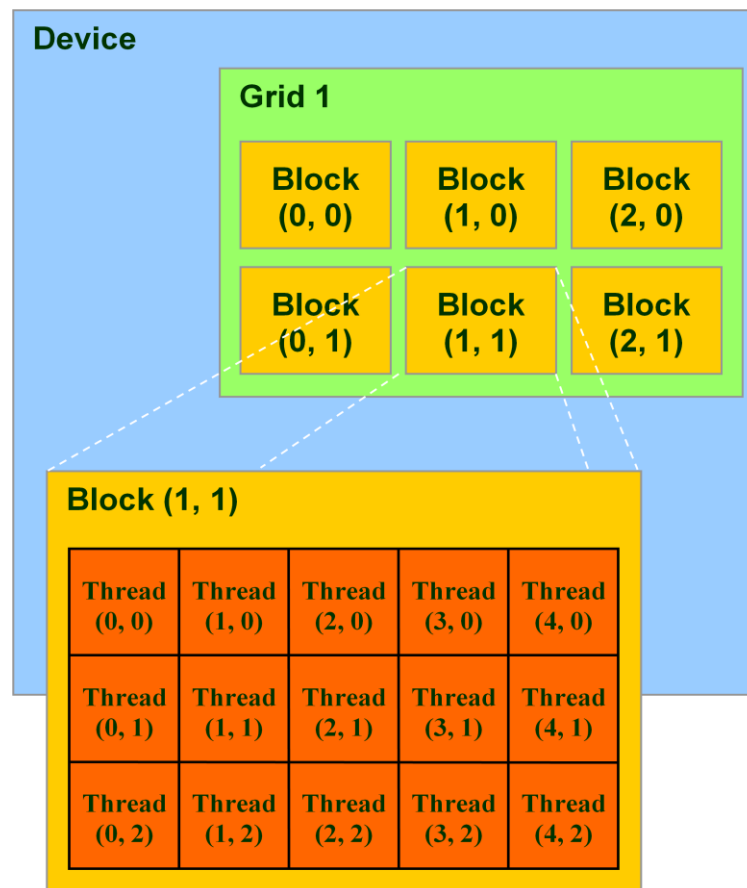


Figure 4.2 Hierarchy of threads in CUDA (NVIDIA Corporation, November 2010)

For convenience, **threadIdx** variable on CUDA is a built-in 3-component vector, so that threads can be identified via this variable. This provides a natural way to map data on memory and invoke computation across the elements in a domain such as a vector, matrix, or volume. There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same processor core and must share the limited memory resources of that core. On current GPUs, a thread block may contain up to 1024 threads (NVIDIA Corporation, November 2010).

Blocks are organized into a one-dimensional or two-dimensional *grid* of thread blocks as illustrated by Figure 4.3. The number of thread blocks in a grid is usually defined by the size of the data being processed due to the limitation to the number of threads per block.

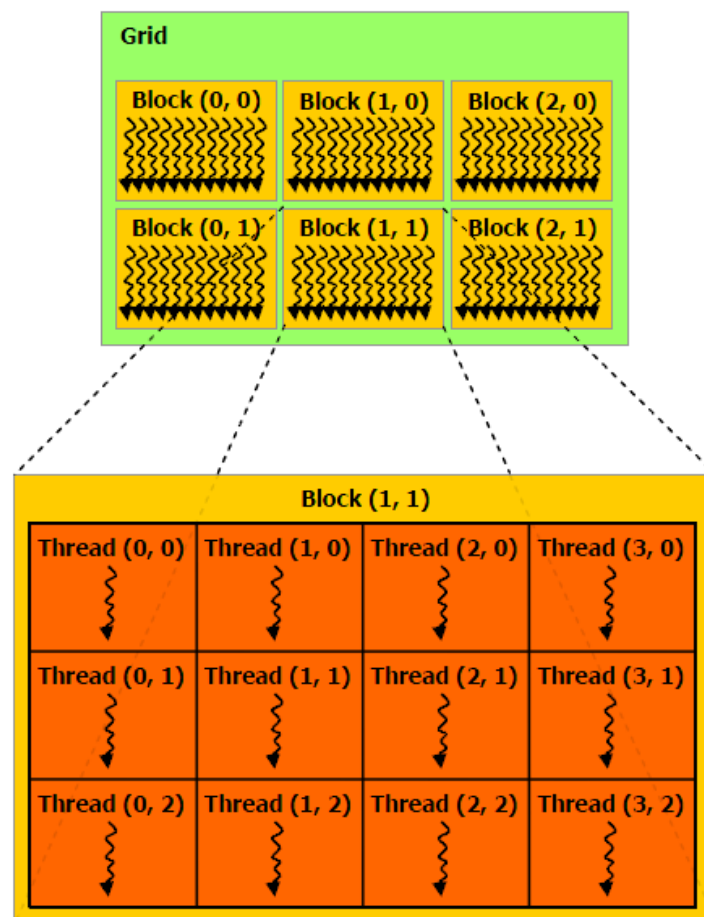


Figure 4.3 Grid of Thread Blocks (NVIDIA Corporation, November 2010)

A kernel is executed by a grid (as illustrated on Figure 4.4). The size of the grid and the thread-blocks are determined by the programmer, according to the size of the data being operated on and to the complexity of the algorithm, at kernel launch time. While threads from different blocks operate independently; threads in a thread block can share data through shared memory and synchronize their execution. Each thread-block in a grid has its own unique identifier and each thread has a unique identifier within a block. Using a combination of block-id and thread-id, it is possible to distinguish each individual thread running on the entire device. Only a single grid of thread blocks can be launched on the GPU at once, and the hardware limits on the number of thread blocks and threads vary across different GPU architectures.

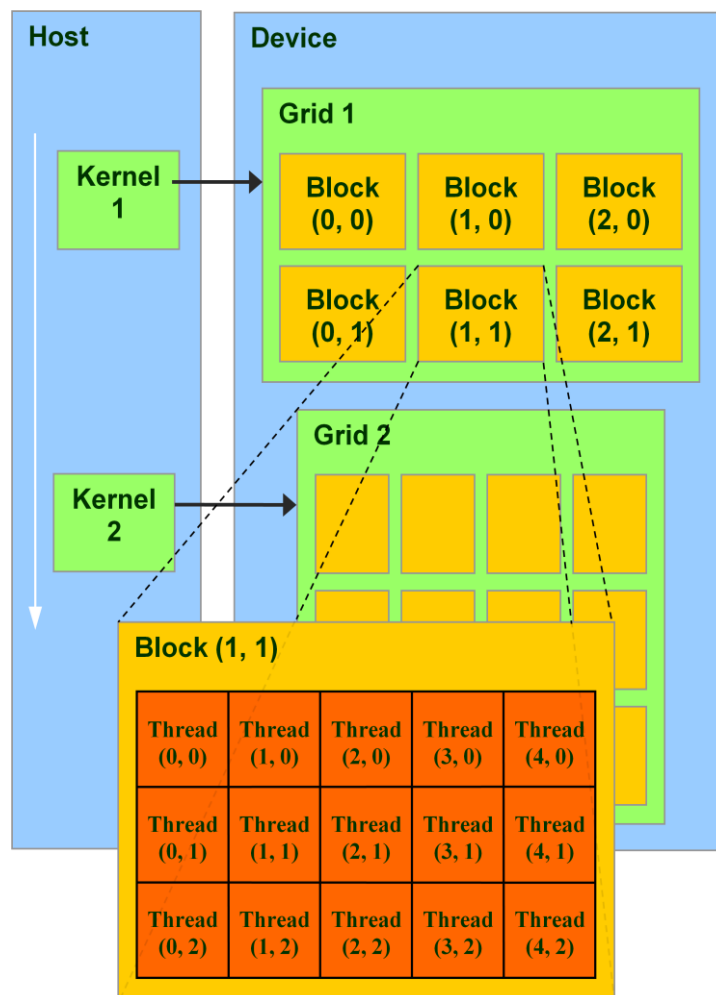


Figure 4.4 Kernel execution and thread model (NVIDIA Corporation, November 2010)

4.2.3 Memory Model

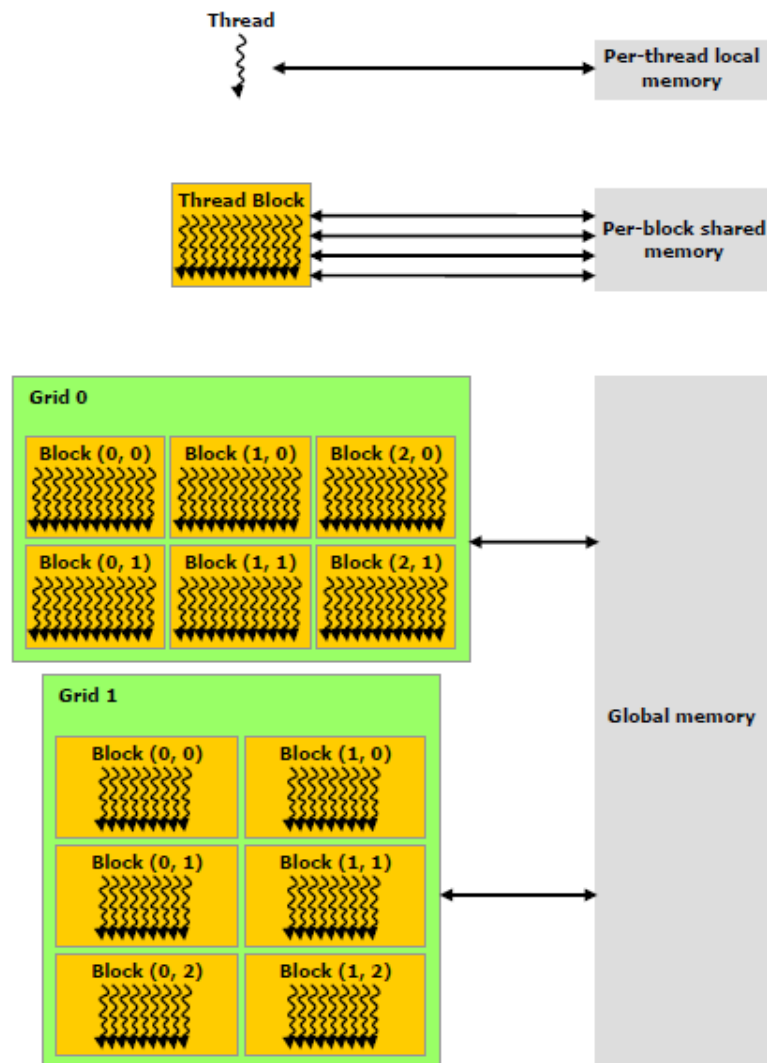


Figure 4.5 Memory hierarchy (NVIDIA Corporation, November 2010)

CUDA threads may access data from multiple memory spaces during their execution as illustrated by Figure 4.5. Each thread has private local memory. Also each thread block has shared memory visible to all threads of the block and with the same lifetime as the block. All threads have access to the same global memory. There are also two additional read-only memory spaces accessible by all threads; the constant and texture memory spaces.

The global, shared, constant and texture memory spaces are optimized for different memory usages. Appropriate use of these memory spaces can have significant performance implications for CUDA applications. The performance characteristics and restrictions of memory spaces are shown on Table 4.1 below:

Table 4.1 Memory accessibility and latency - *Cached only on devices of compute capability 2.x (NVIDIA Corporation, August 2010)

Memory	Location on/off chip	Cached	Access	Scope	Lifetime	Penalty
Register	On	n/a	R/W	1 thread	Thread	1x
Local	Off	No*	R/W	1 thread	Thread	100x
Shared	On	n/a	R/W	All threads in block	Block	1x
Global	Off	No*	R/W	All threads + host	Host allocation	100x
Constant	Off	Yes	R	All threads + host	Host allocation	1x
Texture	Off	Yes	R	All threads + host	Host allocation	1x

With respect to Table 4.1 local and global memories are located off-chip and accessing to these spaces are 100 times slower. On the other hand, although they are located off-chip; accessing constant and texture memory spaces are faster due to caching. Another point from this table is accessibility of memory spaces by threads. According to the Table 4.1 each thread can:

- Read/Write **per-thread** registers
- Read/Write **per-thread** local memory
- Read/Write **per-block** shared memory
- Read/Write **per-grid** global memory
- Read only **per-grid** constant memory
- Read only **per-grid** texture memory

Accessibility of memory spaces are shown in Figure 4.6 below.

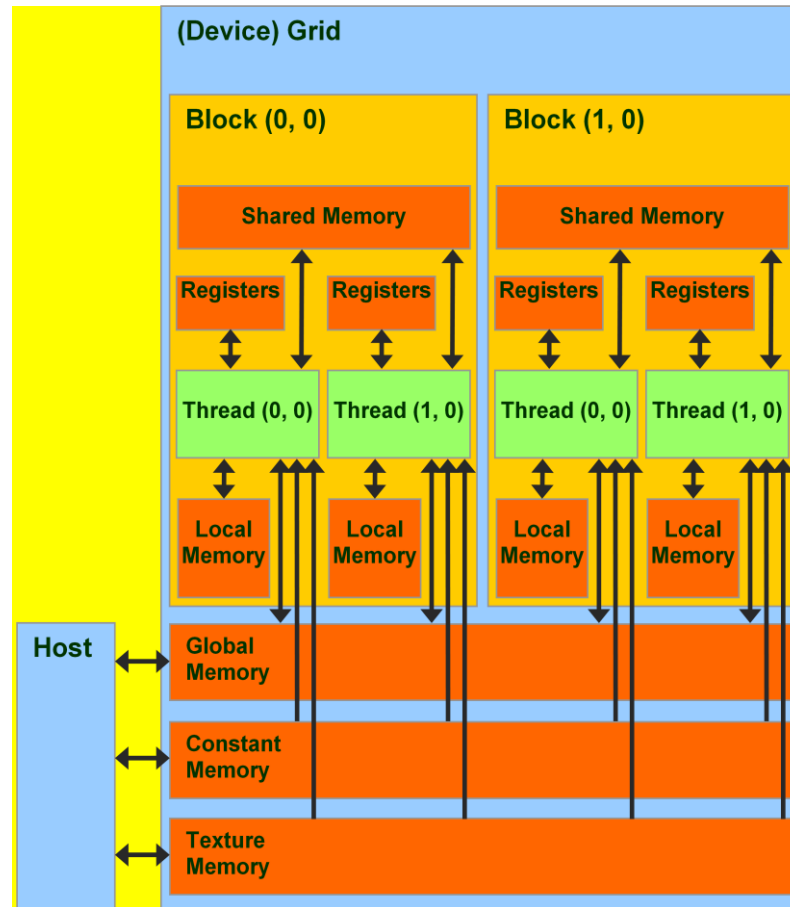


Figure 4.6 Memory Access (NVIDIA Corporation, November 2010)

4.2.3.1 Global memory

Global memory is accessible from either the host or device threads and has the lifetime of the application. Potentially 100x slower than register or shared memory because the global memory resides off-chip and space is not cached, so it is important to follow the right access pattern to get maximum memory bandwidth which has a direct impact to performance.

4.2.3.2 Local Memory

Local memory is only accessible by the threads and has the lifetime of the thread. Actually, local memory is a memory abstraction that implies "local in the scope of each thread". It resides in global memory that is allocated by the compiler and

delivers the same performance as any other global memory region which is 100x slower than register or shared memory.

The variables placed by the compiler to local memory are, arrays which are not indexed with constant quantities, large structures that would consume too much register space and any variable if the kernel uses more registers than available.

4.2.3.3 Shared memory

Shared memory is accessible by any thread of the block from which it was created and has the lifetime of the block. Because it is on-chip, the shared memory space is much faster than the local and global memory spaces. In fact, for all threads of a warp, accessing the shared memory can be as fast as a register when there are no bank conflicts or when reading from the same address. Threads belonging to the same thread block can co-operate with each other, by using shared memory.

4.2.3.4 Registers

Registers are only accessible by threads and have a same lifetime with the thread. They are the fastest form of memory on the multi-processor. Simple scalar variables are placed into registers.

4.2.3.5 Constant Memory

Constant memory is accessible from either the host or device threads and has the lifetime of the application. The constant memory space is cached so a read from constant memory costs one memory read from device memory only on a cache miss. For all threads of a warp, reading from the constant cache is as fast as reading from a register as long as all threads read the same address.

4.2.3.6 Texture Memory

Texture memory is accessible from either the host or device threads and has the lifetime of the application. The texture memory space is cached so a texture sampling costs one memory read from device memory only on a cache miss. Texture memory also offers different addressing modes, as well as data filtering, for some specific data formats.

4.3 CUDA Optimization Strategy

Many type of approaches that can be used to optimize performance on GPU, but especial for CUDA, there are three types of optimization strategy:

- Optimization of instruction usage to achieve maximum instruction throughput
- Optimization of memory usage to achieve maximum memory throughput
- Optimization of parallel execution to achieve maximum utilization

4.3.1 Instruction Throughput

To maximize instruction throughput the programmer should:

- Minimize the use of arithmetic instructions with low throughput, for example single-precision instead of double-precision
- Minimize divergent warps caused by control flow instructions.
- Reduce the number of instructions

4.3.1.1 Arithmetic Instructions

A multiprocessor takes 8 clock cycles for single-precision 32 bit floating-point add, multiply, and multiply-add, integer add, bitwise operations, compare and type conversion instructions on GPU's with compute capability 1.x (NVIDIA Corporation, August 2010).

Integer division and modulo operation are particularly more expensive and should be avoided if possible or replaced with bitwise operations whenever possible

GPU's with compute capability 1.3 has only one double precision floating point unit (FPU) per multiprocessor (SM), is shared by all the threads on SM, whereas there are 8 single precision FPUs. So $\sin f(x)$, $\cos f(x)$, $\tan f(x)$, $\text{sincos} f(x)$ and other double precision operations deliver 8x worse performance than with single precision (NVIDIA Corporation, August 2010).

4.3.1.2 Control Flow Instructions

Threads within a warp execute the same instruction. Thus, in case of flow control instructions (if, switch, do, for, while), threads in a warp may follow different execution paths (divergence) causing significant decline on the effective instruction throughput. In this situation hardware serializes the different executions paths, increasing the total number of instructions executed for this warp. So, if we have two divergent paths within a warp, the two will be serialized, entire warp executing both. When all the different execution paths have completed, the threads converge back to the same execution path. That is where the performance penalty comes from, if flow diverges within a warp. To obtain best performance in cases where the control flow depends on the thread ID, the controlling condition should be written so as to minimize the number of divergent warps (NVIDIA Corporation, August 2010).

4.3.1.3 Memory Instructions

When accessing local or global memory, there are 400 to 600 clock cycles of memory latency. Much of this global memory latency can be hidden by the thread scheduler if there are sufficient arithmetic intensity (independent arithmetic instructions that can be issued) while waiting for the global memory access to complete (NVIDIA Corporation, August 2010).

4.3.2 Memory Bandwidth

GPUs offer high bandwidth throughput. With respect to memory resources, each GPU multiprocessor contains a set of dedicated registers, a store of read-only constant and texture cache, and a small amount of shared memory. These memory types are shared between the individual processors of a multiprocessor. In addition to these memory types, threads evaluated by a processor may also access the GPU's larger, and comparatively slower, global memory. Therefore, programmers should be careful while designing algorithm and organizing memory accesses because wrong usage of these memory spaces directly affects the performance. The access time penalties of different memory spaces are shown on Table 4.2.

Table 4.2 Access time penalties of different memory spaces on GPU (NVIDIA Corporation, August 2010)

Memory	Penalty
register	1x
local	100x
shared	1x
global	100x
constant	1x

The first step in maximizing overall memory throughput for the application is to minimize data transfers with low bandwidth by minimizing data transfers between the host and the device. Since off-chip device memories are of much higher latency and lower bandwidth than on-chip memory, memory accesses to them should be minimized.

Shared memory can be seen as a user-managed cache. A typical programming pattern is to cache data coming from device memory into shared memory; in other words, to have each thread of a block:

- Load data from device memory to shared memory

- Synchronize with all the other threads of the block so that each thread can safely read shared memory locations that were written by different threads
- Process the data in shared memory
- Synchronize again if necessary to make sure that shared memory has been updated with the results
- Write the results back to device memory

4.3.2.1 Data Transfers between Host and Device

Because of the overhead associated with each transfer, instead of transferring small portions of data separately, batching many small transfers into a single large transfer always performs better.

4.3.2.2 Global Memory Accesses

The task of effectively hiding the global memory access latency and managing the memory hierarchy is very crucial for obtaining maximal performance from the GPU.

The global memory space is not cached, so it is all the more important to follow the right access pattern to get maximum memory bandwidth, especially given how costly accesses to device memory are.

First, the device is capable of accessing device memory via 32, 64, or 128 byte memory transactions (NVIDIA Corporation, August 2010). When a warp executes an instruction that accesses global memory, it unites the memory accesses of the threads within the warp into one or more of these memory transactions, depending on the size of the word accessed by each thread. So the structure layout on device memory must be aligned to their size (or multiple of their size) in order to achieve memory transactions without latency. Because, global memory bandwidth is used most efficiently when the simultaneous memory accesses by threads in a half-warp can be coalesced into a single memory transaction. Coalescing can be maximized by

redesigning structure to most optimal access patterns and using data types that meet the size and alignment.

4.3.2.3 Local Memory

Like the global memory space, the local memory space is not cached, so accesses to local memory are as expensive as accesses to global memory and are subject to the same requirements for memory coalescing as described at Section 4.3.2.2.

4.3.2.4 Constant Memory

The constant memory is placed off-chip but space is cached so a read from constant memory costs one memory read from device memory only on a cache miss, otherwise it just costs as one read from its cache.

4.3.2.5 Texture Memory

The texture memory space is placed off-chip but space is cached so a texture fetch costs one memory read from device memory only on a cache miss, otherwise it just costs one read from the its cache.

4.3.2.6 Shared Memory

The shared memory space is much faster than the local and global memory spaces. In fact, for all threads of a warp accessing the shared memory is as fast as accessing a register as long as there are no bank conflicts between the threads, as detailed below.

Shared memory is divided into equally-sized memory modules, called banks, which can be accessed simultaneously. So, any memory transaction request made of n addresses that fall in n distinct memory banks can be serviced simultaneously, yielding an effective bandwidth that is n times as high as the bandwidth of a single

module. It is therefore important to understand how memory addresses map to memory banks in order to schedule the memory requests so as to minimize bank conflicts.

4.3.2.7 Registers

Generally, accessing a register has no latency since it doesn't require an extra clock cycle, but delays may occur due to register read-after-write dependencies and register memory bank conflicts.

4.3.3 Occupancy

One of the key challenges in algorithmic design for GPGPUs is to keep all processing elements busy. In other words, to ensure high utilization (occupancy) of resources and provide more parallel work is dispatched than the stream processors available. Using latency-hiding techniques, a processor waiting on a memory accessing can thus simply switch context to another dispatched work unit which has load its necessary data from memory.

The multiprocessor occupancy is the ratio of active warps to the maximum number of warps supported on a multiprocessor of the GPU. Maximizing the occupancy can help to cover latency during global memory loads. The occupancy is determined by the amount of shared memory and registers used by each thread block. Because of this, programmers need to choose the size of thread blocks with care in order to maximize occupancy. Each multiprocessor on the device has a set of N registers available for use by CUDA program threads. These registers are a shared resource that is allocated among the thread blocks executing on a multiprocessor. The CUDA compiler attempts to minimize register usage to maximize the number of thread blocks that can be active in the machine simultaneously. If a program tries to launch a kernel for which the registers used per thread times the thread block size is greater than N , the launch will fail.

It is important to note that two key resources of the SM, namely the shared memory and the register file, are shared by the thread-blocks that are concurrently active on the SM. For example, if each SM has 16KB of shared memory and each thread-block requires 8KB of shared memory, then no more than 2 thread blocks can be concurrently scheduled on the SM, as it is seen in Figure 4.7.

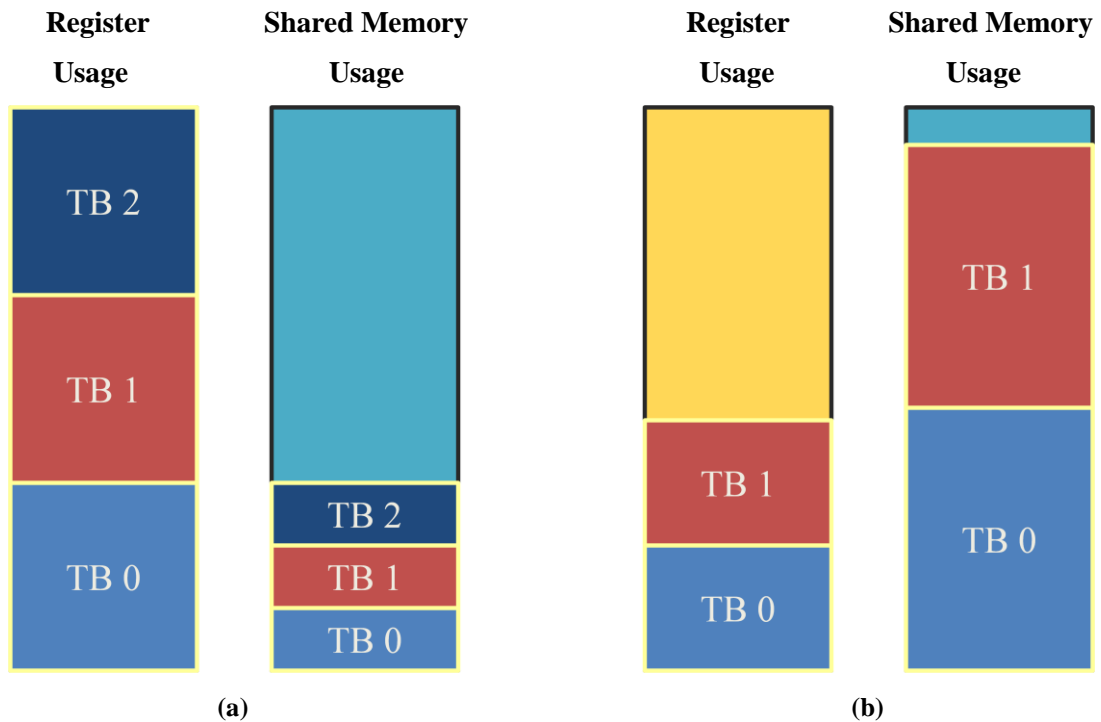


Figure 4.7 GPU Resources are not utilized properly.

From Figure 4.7, in (a) we are wasting the available shared memory space and there is only 3 thread block can be concurrently active because of heavily register usage. We should move some variables from registers to shared memory to balance resource usage. Similarly in (b) resources wasted because of high shared memory usage and we should reduce shared memory usage by threads in order to increase occupancy.

CHAPTER FIVE

LEMMATIZATION ON GPU

5.1 Lemmatization Algorithm on CUDA

While working on CUDA the methods applied to the development should include the following:

- 1) Minimize data transfer with global memory
- 2) Work on faster accessible memory units
- 3) Accessing of global memory should be coalesced as much as possible
- 4) Avoid branch divergence within a CUDA warp
- 5) Use resources of GPU efficiently to avoid limitations of hardware

In order to work efficient under these constraints, we had to change our trie structure. While we were optimizing our code through CUDA our guidelines were:

- 1) Get rid of pointers. Working with pointers on GPU is not efficient.
- 2) Minimize memory read/writes.
- 3) Minimize divergent (if-else, for, while) blocks.
- 4) Minimize memory usage of variables.
- 5) Do load/store works on faster memory units and later store result in slower units.
- 6) Reduce instructions and complexity as possible.

5.1.1 Redesigning Structure

First of all we changed our trie structure to node array (namely, *array of structs*) in order to get rid of pointers. Instead of storing each child's pointer in parent node we stored the child's index at node array. And we inserted our nodes on array by traversing tree with two different approaches, breadth-first and pre-order traversal basis, in the cause of memory access coalescing (discussed in Section 4.3.2.2).

After that we changed our look-up algorithm with regard to the changes in structure. The previous and latter structures can be seen in Figure 5.1.

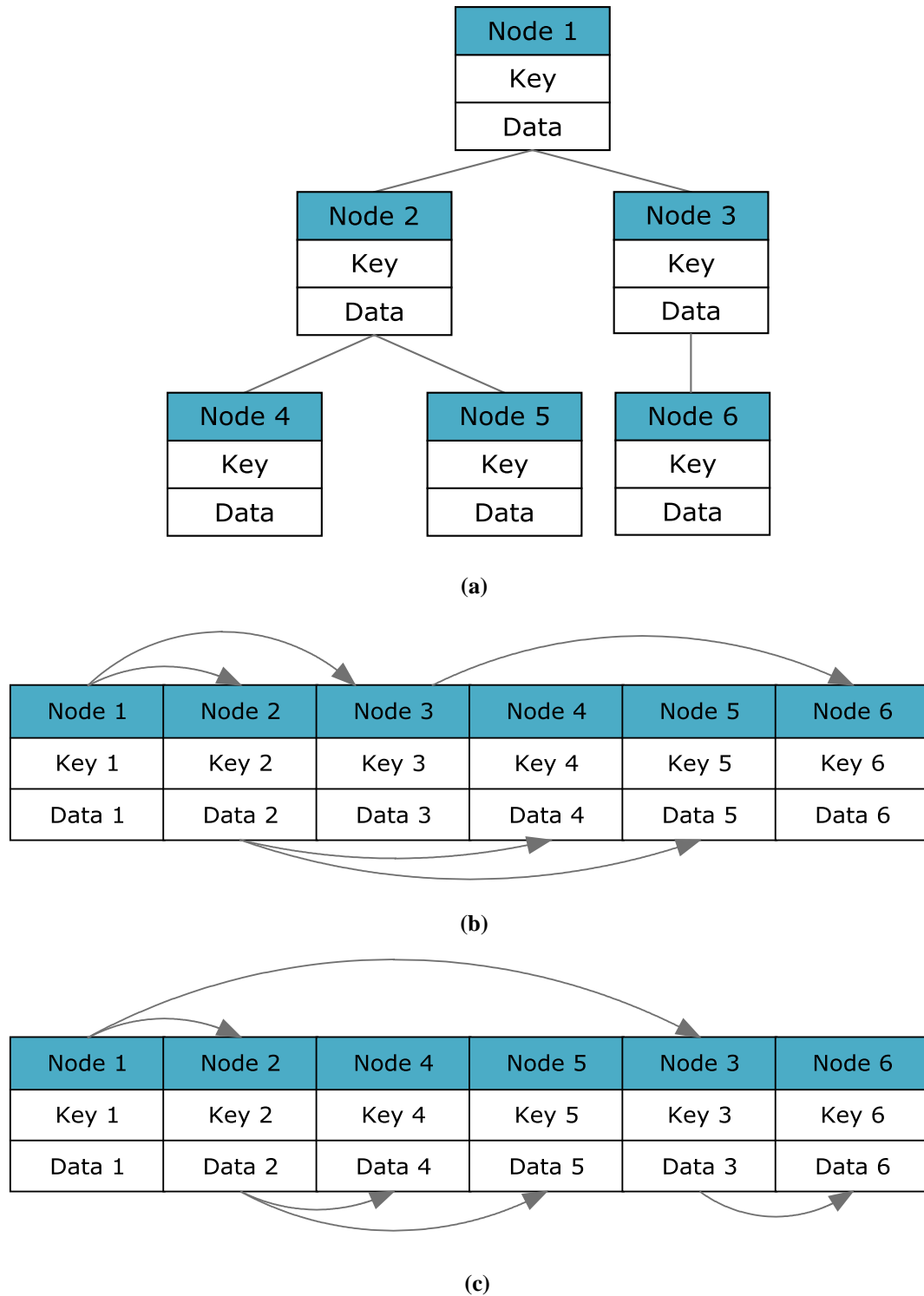


Figure 5.1 (a) Trie Structure. (b) Array of nodes structure with nodes placed via breadth-first traversal on trie. (c) Array of nodes structure with nodes placed via pre-order traversal on trie.

Arrays of structures (AOS) keep things nicely organized but are generally bad for performance in data parallel computation. When the structure is laid out in memory, the compiler will produce interleaved data, in the sense that all the structures will be contiguous but there will be a constant offset between a structure instance and the same element of the following instance. This offset particularly depends on the structure definition. To make sure SIMD operations can work efficiently on data, they shall be allocated in continuous memory space. So the best bet for performance is to design software around structures of arrays (SOA).

In GPU, global memory is accessed in chunks depending on to memory bus. If we don't use whole chunk the bandwidth is wasted (NVIDIA Corporation, August 2010). If we look at the memory layout in global memory, the AOS layout would have all the node's contents together, whereas in the SOA layout we would have all the keys (required data) together in RAM. So in theory, the SOA layout would be better performing because when we access the key data, we will get more data in chunk since size of key is smaller than whole struct.

For example; assuming a chunk size of 32 bytes and we have 16 nodes where a node consists a *key* with 2 bytes and a *data* with 2 bytes; if the AOS algorithm would want to access key of node 1, and then to the key of node 9; this request will cause a chunk miss, causing the processor to fetch in node 9 into the chunk by a second read. On the other hand with SOA algorithm, all 16 keys can be read into chunk by one read, providing a performance boost.

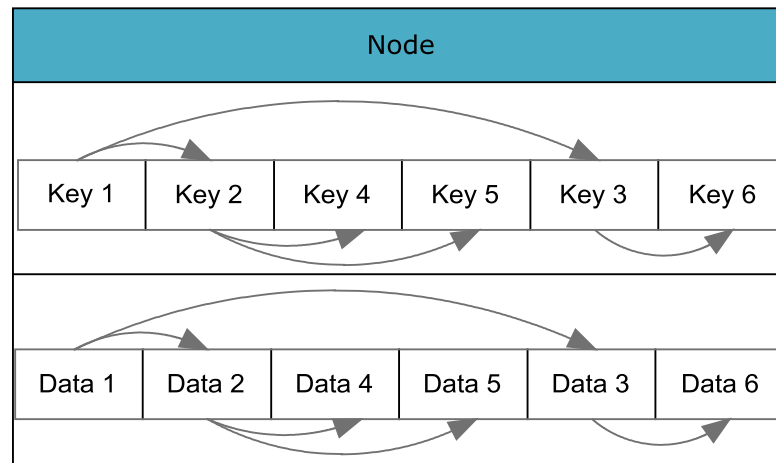


Figure 5.2 Node (struct) of arrays structure

AOS is also faster when all the data in the chunks are aligned to multiples of 32/64/128 bytes but becomes slower when it does not. The takeaway from all this is the layout of our data affects the running speed by a large amount, but it's also important to write small tests to validate whether SOA or AOS better. In our case SOA performed better than AOS. The results are written on evaluation part, Section 6.2 of the thesis.

To decide which structure performs best we developed 9 different version of our lemmatizer each is working with distinct structure:

- **Sequential algorithm with tree struct**

A sequential, CPU-based version of our lemmatizer which uses a radix trie.

- **Parallel array of structs algorithm**

A data parallel version of our algorithm which uses a struct array instead of trie. But look-up algorithm is similar to radix search. This algorithm uses 12 registers, 42 bytes local memory and 48 bytes shared memory on GPU with 256 threads per block.

- **Parallel arrays algorithm**

A data parallel version of our algorithm which uses separate arrays instead of struct array. We defined each property of struct as a separate array. Look-up algorithm is similar. This algorithm uses 15 registers, 42 bytes local memory and 96 bytes shared memory on GPU with 256 threads per block.

- **Parallel struct of arrays algorithm**

A data parallel version of our algorithm which uses a structure of arrays instead of an array of structures. We defined each property of struct as a separate array in 1 unique struct. Look-up algorithm is similar. This algorithm uses 15 registers, and 42 bytes local memory 48 bytes shared memory on GPU with 256 threads per block.

- **Parallel array of structs algorithm with compact (smaller) nodes**

A data parallel version of our algorithm which uses a struct array instead of trie. But this time structs are smaller because we removed Data, MasterData, MasterKey due to fact that they are not necessary for lemmatizing.(these properties was added to structs for WSD and query/document expansion purposes). Look-up algorithm is similar to radix search. This algorithm uses 13 registers, 42 bytes local memory and 48 bytes shared memory on GPU with 256 threads per block.

- **Parallel arrays algorithm with compact (smaller) arrays**

A data parallel version of our algorithm which uses a struct array instead of trie. Data, MasterData and MasterKey arrays are removed. Look-up algorithm is similar to radix search. This algorithm uses 15 registers, 42 bytes local memory and 88 bytes shared memory on GPU with 256 threads per block.

- **Parallel struct of arrays algorithm with compact (smaller) nodes with compact (smaller) nodes inserted from trie via pre-order traversal basis**

A data parallel version of our algorithm which uses a structure of arrays instead of an array of structures. We defined each property of struct as a separate array in 1 unique struct but removed Data, MasterData and MasterKey properties. Look-up algorithm is similar. This algorithm uses 15 registers, 42 bytes local memory and 48 bytes shared memory on GPU with 256 threads per block.

- **Parallel struct of arrays algorithm with compact (smaller) nodes inserted from trie via pre-order traversal basis and also exploits shared memory**

A data parallel version of our algorithm which uses a structure of arrays instead of an array of structures. We defined each property of struct as a separate array in 1 unique struct but removed Data, MasterData and MasterKey properties. In addition; we also carried some variables into shared memory in order to reduce global memory read/writes. Look-up algorithm is similar. This algorithm uses 16 registers, 21 bytes local memory and 3888 bytes shared memory on GPU with 256 threads per block.

- **Parallel struct of arrays algorithm with compact (smaller) nodes inserted from trie via breadth-first traversal basis and also exploits shared memory**

A data parallel version of our algorithm which uses a structure of arrays instead of an array of structures. But this time, we preferred placing structs on array from trie via breadth-first basis. This approach visits the elements level-by-level. So we inserted all the nodes on current level of trie before we proceeded to sub levels. With this algorithm, the only difference is layout of dictionary on memory. We defined each property of struct as a separate array in 1 unique struct but removed Data, MasterData and MasterKey properties. In addition; we also carried some variables into shared memory in order to reduce global memory read/writes. Look-up

algorithm is similar. This algorithm uses 16 registers, 21 bytes local memory and 3888 bytes shared memory on GPU with 256 threads per block.

5.1.2 Occupancy

In our algorithm, trie needs 12.31MB space which can only reside on global memory of GPU. So we had to achieve coalesced accessing and warp occupancy as much as possible to hide latency of memory transactions. In order to achieve an occupancy ratio of 1, we redesigned our algorithm regard to GPU specifications (can be seen on Appendix 2). The specifications of our GPU as follows:

- Total global memory : 947 MB
- Shared memory per processor : 16 KB
- Warp size : 32
- Max. threads per block : 512
- Total constant memory : 64 KB
- Clock rate : 1210000KHz
- Multiprocessors on device : 6
- Multicores on each processor : 8
- Max count of threads in each processor : 1024
- Max count of register : 16384

Regard to this specifications we should keep shared memory usage by each thread block under 16KB considering the fact that each multi-processor has 16KB shared memory unit. Also there is a limit of 16384 registers per multiprocessor. So we should organize our shared memory, register and thread usage considering these limitations to prevent performance comedown.

Our kernel uses 16 Registers per each thread. So we can map maximum of $16384/16 = 1024$ threads on a multiprocessor without decreasing warp occupancy. If we select our thread number per block 256 then we will have $1024/256=4$ blocks of threads each needs $256*16=4096$ registers. To fit in 4 blocks of threads in a

multiprocessor without reducing occupancy ratio; we should limit our shared memory usage on each block to maximum of 4KB (since $4\text{KB} * 4 \text{ block} = 16 \text{ KB}$) and on each thread to $4\text{KB} / 256 = 16 \text{ bytes}$. So in final analysis, considering our kernel uses 16 registers per thread if we select our thread number per block 256 we should use 16 bytes of shared memory per thread or 4KB of total to achieve full warp occupancy.

Our resource usage and occupancy measurements can be seen on Figure 5.3, Figure 5.4 and Figure 5.5.

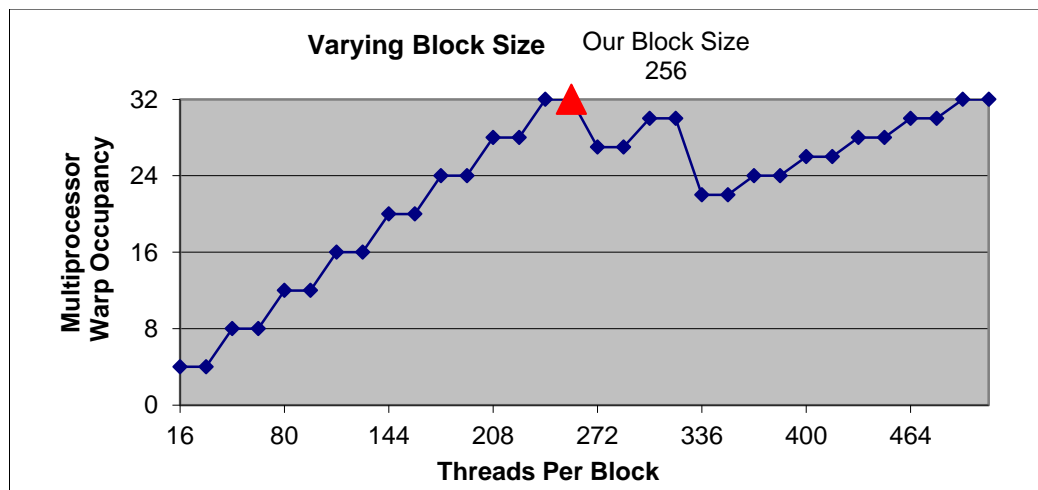


Figure 5.3 Selected threads per block to achieve full occupancy. Red triangle shows our Block size

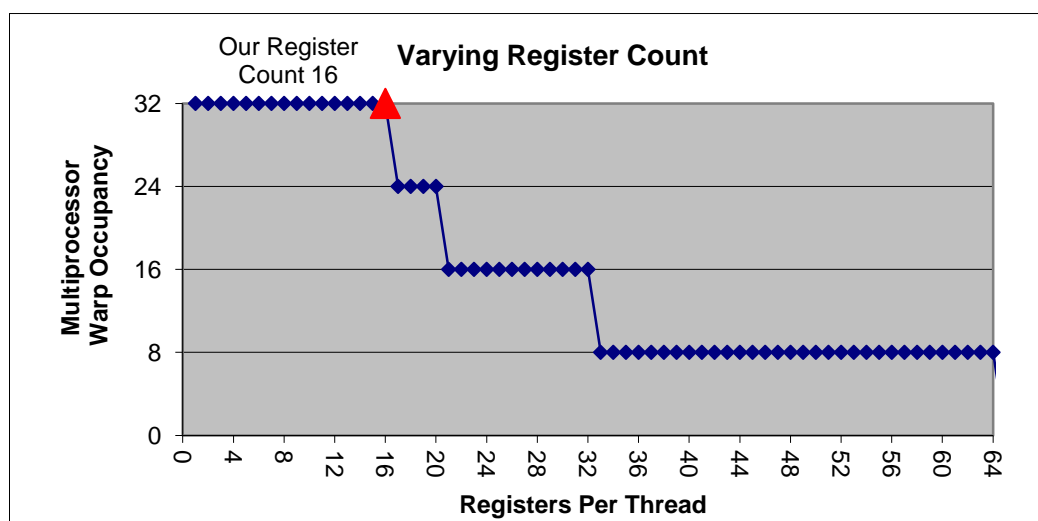


Figure 5.4 Optimized register usage of kernel to achieve full occupancy. Red triangle shows our register usage per thread.

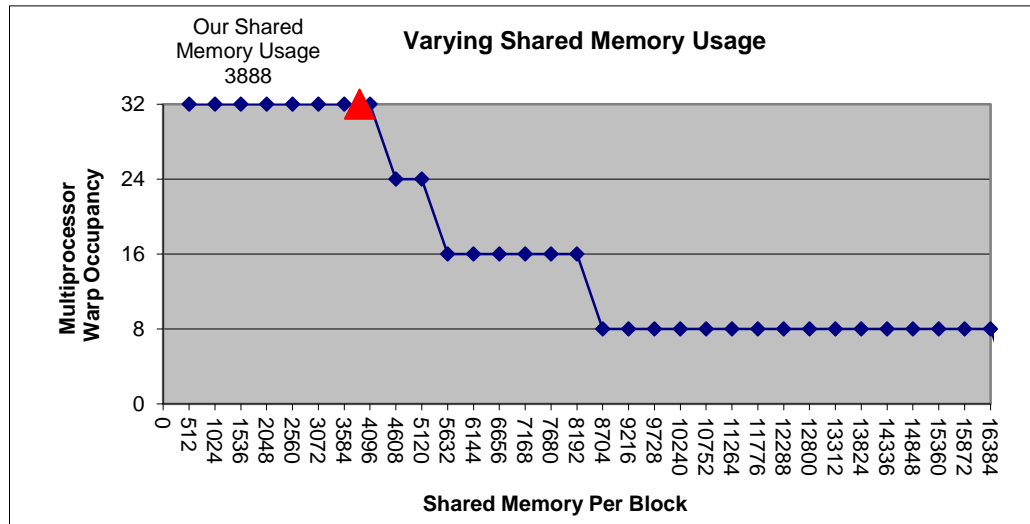


Figure 5.5 Optimized shared memory usage to achieve full occupancy. Red triangle shows our register usage per thread.

These figures tell us that we have full utilization of resources and achieve a warp occupancy ratio of 1. We could also use 512 threads per block and get full occupancy (32 threads) but in order to do that we should lower the shared memory usage by threads. Because if we increase thread count without limiting their shared memory usage we will have a shared memory overflow which causes a performance decline.

CHAPTER SIX

EVALUATION

6.1 Test Data and Measurement Method

6.1.1 Test Data

To measure our lemmatizer's accuracy and performance we have used the recently constructed *Milliyet* dataset (the news articles and columns of 5 years, 2001–2005, from the Turkish daily newspaper *Milliyet* (<http://www.milliyet.com.tr>)) for Turkish along with the TREC-style query and relevance judgments set (Can, Kocerberber, Balcik, Kaynak, Ocalan, & Vursavas, 2008). The dataset includes 408,305 documents which contains 95.5 million words (each document contains 234 words on the average), with 72 ad-hoc queries and 33 assessors. The query set provided as a set of words that describes a user information need with three fields: topic (a few words), description (one or two sentences), and narrative (more explanation). We used the queries on the description field in our tests which includes 72 queries consisting 936 words with 13 terms average and 1.74 stop words, on average (queries can be seen at Appendix 5).

6.1.2 Measurement

Precision and recall are the common evaluation measures in information retrieval. Most of the other measures are derived from them.

- a) Recall is the measure of the ability of a system to present all relevant items.

$$\text{Recall} = \frac{\text{Number of relevant items retrieved}}{\text{Number of relevant items in collection}}$$

- b) Precision is the measure of the ability of a system to present only relevant items.

$$\text{Precision} = \frac{\text{Number of relevant items retrieved}}{\text{Total number of items retrieved}}$$

Where, retrieved is number of documents retrieved by a search of query and relevant is total possible relevant documents within a given query.

Retrieval tasks whose results are a ranked list of documents can be evaluated by the trec_eval program. “trec_eval” was written by Chris Buckley (It is available from the TREC website at http://trec.nist.gov/trec_eval). We used the trec_eval package version 8.1 for obtaining the effectiveness measures. An evaluation report for a run evaluated by “trec_eval” gives a report with the measurements following (trec_eval output can be seen at Appendix 1):

Table 6.1 Trec_eval measurement types

Name	Description
num_ret	Total number of documents retrieved over all queries
num_rel	Total number of relevant documents over all queries
num_rel_ret	Total number of relevant documents retrieved over all queries
map	Mean Average Precision (MAP)
gm_map	Average Precision. Geometric Mean, $q_score = \log(\text{MAX}(\text{map}, .00001))$
Rprec	R-Precision (Precision after R (= num-rel for topic) documents retrieved)
bpref	Binary Preference, top R judged nonrel
recip_rank	Reciprocal rank of top relevant document
iprec@recall_N	Interpolated Recall - Precision Averages at N recall
P@N	Precision after N docs retrieved

6.2 Evaluation of Lemmatizer Accuracy

Firstly we wanted to see how accurate our lemmatizer on a small set. To achieve this task, we manually lemmatized words on queries provided by dataset (72 Queries, 936 words) via “Büyük Türkçe Sözlük” (Turkish Language Association’s Grand Dictionary) in order to create base lemmas for measurement. After that we applied

our lemmatizer on same query set. Then we compared the base lemmas with our lemmatizer's output for each query. Table 6.2 shows a small fraction of the output, the rest of this table can be seen at Appendix 6.

Table 6.2 Lemmatization of query set

Query	Lemmatized Query	Accuracy	Summary
Kuş gribi nedir, nasıl bulaşır, belirtileri nelerdir sorularına cevap olabilecek dokümanlar.	kuş grip bulaş belirti soru cevap olabil doküman	100%	
Türkiye'nin Avrupa Birliği'ne tam üyelik sürecinde Kıbrıs sorununu ele alan bir doküman.	türkiye avrupa birlik tam üyelik süreç kıbrıs sorun ele alan doküman	90.91%	The lemma of "ele" must be "el" (hand) but our lemmatizer returns "ele"+ "(mek)" (to eliminate) And the lemma of "alan" must be "al"+ "(mak)" (to take) But our lemmatizer return "alan"(region)
Türkiye'de üniversiteye giriş sınavının gençler üzerindeki etkileri, gençlerin ve kamuoyunun bu sınav için düşündükleri.	türkiye üniversite giriş sınav genç etki genç kamuoyu sınav düşün	100%	
Güney Asya'yı 26 Aralık 2004'te vuran büyük Tsunami faciası ve bu facianın sonuçları.	güney asya 26 aralık 2004 vur büyük tsunami facia facia sonuç	100%	
Mavi akımın ulusal enerji politikamızdaki yeri, ekonomik maliyeti	mavi akım ulusal enerji politika yer ekonomik maliyet	100%	
Büyük bir bölümü deprem bölgesi olan Türkiye'de deprem öncesi alınan tedbirler nelerdir?	büyük bölüm deprem bölge ol türkiye deprem önce alın tedbir	100%	
Türk Silahlı Kuvvetleri ile PKK arasında meydana gelen çatışmalar	türk silahlı kuvvet pkk arası meydan gelen çatışma	100%	

The queries consist of total 936 words. Eliminating stop words from queries leaves us 786 words and our lemmatizer's total accuracy is with 764 correct lemmas equals to %97.201 (764 correct lemmas / 786 words).

Of course this evaluation was not sufficient enough to make a decision about our lemmatizer's effectiveness. So we decided to build an information retrieval (IR) system and observe our lemmatizer's impact on retrieval process.

While creating our test environment, we didn't want to deal with the development of an IR system from scratch; so instead, we used "Lucene" which is an open source IR software library, created by Doug Cutting (It is available at <http://lucene.apache.org/java/docs/index.html>). Because "Lucene" offers users full text indexing and searching capability along with:

- ranked searching (best results returned first)
- Many query types: phrase queries, wildcard queries, proximity queries, range queries etc.
- fielded searching (e.g., title, author, contents)
- sorting by any field
- multiple-index searching with merged results
- simultaneous update and searching

After we dealt with IR system development problem via "Lucene"; we compared the effects of three different approaches on (Turkish) IR effectiveness on "Lucene":

- a) **NS**: The abbreviation stands for no stemming. This approach uses all words as an indexing term. The retrieval performance of this approach provides a baseline for comparison.
- b) **FPT5**: The abbreviation stands for fixed prefix truncation by length of 5 characters. We simply truncate the words and use the first 5 characters of each

word as its stem; words with less than 5 characters are used as a stem with no truncation. We used this fixed prefix stemmer because it had shown before that it produces good results on Turkish language (Can & others., 2008).

c) **LDB**: This abbreviation stands for our dictionary based lemmatizer.

In this study, we also used a stop words list (stop words list can be seen on Appendix 3) consists of the most frequent words of Turkish language, and some manually added words. Then we applied FPT5 stemmer to these words. So in final case, we generated a stop word list composed of 5 character-length words (our stop words list can be seen on Appendix 4). Later, we used this stop word list to eliminate words, *before* applying the stemmers to them. For this purpose, we first used the FPT5 stemmer to find the appropriate stem, and then we searched the stemmed word in the stop word list.

The indexing information on “Lucene” with different stemmers, using the stop word list, is shown in Table 6.3.

Table 6.3 The indexes created for search engine

	NS	FPT5	LDB	Gain % of FPT5 over NS	Gain % of LDB over NS	Gain % of LDB over FPT5
Indexed Term Count	1679002	283365	69099	83.12%	95.88%	75.61%
Index Size	1584MB	1357MB	1004MB	14.33%	36.62%	26.01%

From table, it means that FPT5 and LDB provide 14.33% and 36.62%, respectively, storage efficiency with respect to NS. The storage size of the index builded with LDB is the most efficient among others.

To encapsulate, our evaluation process can be summarized as follows. First we constructed three different indexes via three different options (listed above) applied to indexing process respectively. Then we applied selected option on the queries in the same way. After that we ran each of the queries on the system using the index

that is created with same option as the query stemmed, and then with the information returned by system, we created a TREC-style output by using the first 1000 results returned. This output allows us to measure the results in “*trec_eval*”. Thus, finally, we measured the IR effectiveness of these three stemming approaches with “*trec_eval*” program and compared them. The measurement results are below on Table 6.4:

Table 6.4 Trec_eval measurement results.

	NS	FPT5	LDB	% of LDB – NS increase	% of LDB – FPT5 increase
num_ret	72000	72000	72000	-	-
num_rel	7510	7510	7510	-	-
num_rel_ret	4136	4870	5424	31.14	11.38
map	0.1904	0.2288	0.2941	54.46	28.54
gm_map	0.0771	0.1148	0.2063	167.57	79.70
Rprec	0.2352	0.2728	0.3356	42.69	23.02
bpref	0.3406	0.4036	0.4144	21.67	2.68
recip_rank	0.5701	0.6688	0.7899	38.55	18.11
P@5	0.4333	0.5278	0.6139	41.68	16.31
P@10	0.4125	0.4847	0.5667	37.38	16.92
P@15	0.4093	0.4630	0.5481	33.91	18.38
P@20	0.3882	0.4375	0.5188	33.64	18.58
P@30	0.3560	0.3981	0.4796	34.72	20.47
P@100	0.2332	0.2737	0.3300	41.51	20.57
P@200	0.1670	0.1976	0.2302	37.84	16.50
P@500	0.0955	0.1123	0.1283	34.35	14.25
P@1000	0.0574	0.0676	0.0753	31.18	11.39
iprec@recall_0.00	0.6273	0.7130	0.8214	30.94	15.20
iprec@recall_0.10	0.3764	0.4515	0.5863	55.77	29.86
iprec@recall_0.20	0.3058	0.3684	0.4913	60.66	33.36
iprec@recall_0.30	0.2527	0.3131	0.4112	62.72	31.33
iprec@recall_0.40	0.2139	0.2538	0.3422	59.98	34.83
iprec@recall_0.50	0.1741	0.2120	0.2749	57.90	29.67
iprec@recall_0.60	0.1328	0.1697	0.2137	60.92	25.93
iprec@recall_0.70	0.1020	0.1260	0.1653	62.06	31.19
iprec@recall_0.80	0.0691	0.0824	0.1103	59.62	33.86
iprec@recall_0.90	0.0433	0.0388	0.0455	5.08	17.27
iprec@recall_1.00	0.0073	0.0012	0.0116	58.90	866.67

To give a judgment on which is the best of these stemming approaches, we should consider the *precision – recall average*, *bpref*, *GM_MAP*, *MAP*, *P@10*, and *P@20* values on Table 6.4.

6.2.1 Precision at N documents

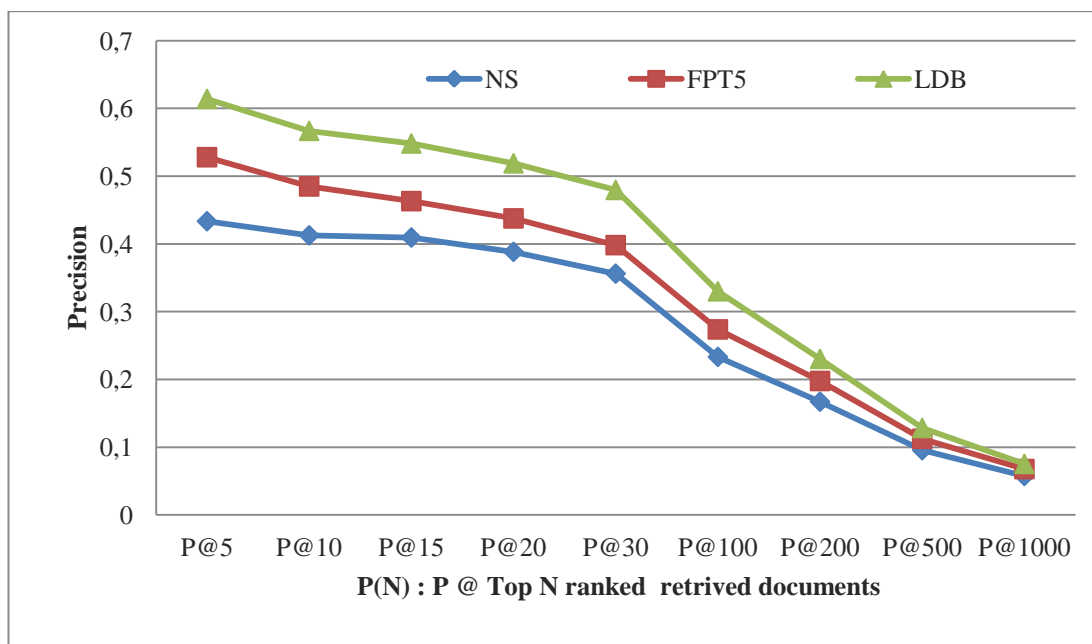


Figure 6.1 Visual presentation of precision at top N ranked retrieved documents

Precision at the top N documents, commonly 10 and 20 documents ($P@10$, $P@20$), are preferred measure because of their simplicity and intuitiveness. The precision computed after a given number of documents have been retrieved reflects the actual measured system performance as a user might see it.

$P@10$ and $P@20$ values of LDB are about 17% and 18.5% higher than that of FPT5, also about 37% and 33.5% higher than that of NS. Due to these observations, our lemmatizer provides better results for first 20 results which are the results which an ordinary user will commonly look only at them.

6.2.2 Precision – Recall Averages

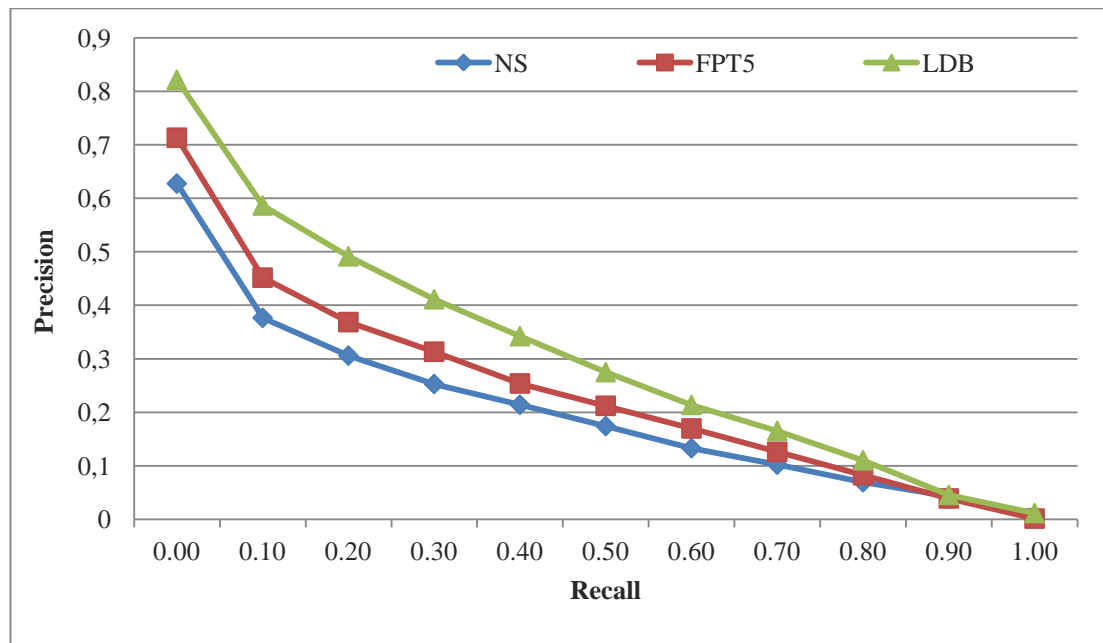


Figure 6.2 Visual presentation of interpolated precision - recall averages

The precision - recall graph (Figure 6.2) is created using the 11 cutoff values from the precision at recall level averages on Table 6.4. Characteristically these graphs slope downward from left to right, enforcing the notion that as more relevant documents are retrieved (recall increases); the more non-relevant documents are retrieved (precision decreases).

This graph is the most commonly used method for comparing systems. Curves closest to the upper right-hand corner of the graph (where recall and precision are maximized) indicate the best performance. The plots of different stemmers are plotted on the same graph and it can be clearly seen that LDB is superior to both other approaches.

6.2.3 Map, Gmap and Rprec

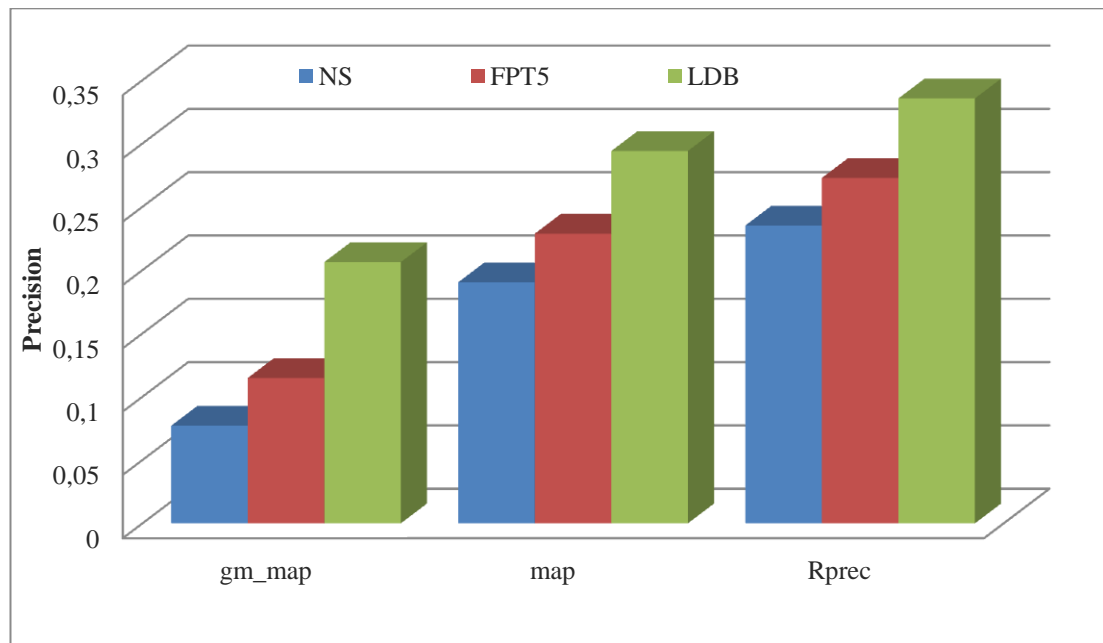


Figure 6.3 Visual representations of gm_map, map and Rprec values

MAP is the mean of the average precision value that reflects the performance over all relevant documents. The measure is not an average of the precision at standard recall levels. Rather, it is the average of the precision value obtained after each relevant document is retrieved. *MAP* is considered as a more reliable measure for effectiveness (Buckley & Voorhees, 2004; Sanderson & Zobel, 2005).

In terms of *MAP* measure, the performance of LBD is 28.54% better than FPT5's performance and has an increase of 54.46% than that of NS. According to the MAP results, FPT5 is obviously dropping behind LDB.

The geometric mean average precision (*GMAP*) measures improvements for low-performing queries. *GMAP* is the geometric mean of per-query average precision, in contrast with *MAP* which is the arithmetic mean. If a run doubles the average precision for topic A from 0.03 to 0.06, while decreasing topic B from 0.3 to 0.27, the arithmetic mean is unchanged, but the geometric mean will show an improvement.

GMAP measures show us that LDB is 79.70% better than FPT5 and 167.57% than NS. Again, LDB is the best effective option.

R-Precision is the precision after R documents have been retrieved, where R is the number of relevant documents for the query. It trivializes the exact ranking of the retrieved relevant documents, which can be particularly useful in TREC where there are large numbers of relevant documents. LDB outpaces FPT5 and NS on this measure with 23.02% and 43.62%, respectively.

6.2.4 *Bpref*

Table 6.4 also shows the performance of NS, FPT5, and LDB in terms of *bpref* and the percentage improvement provided by LDB with respect to NS and FPT5. For easy comparison, *bpref* values of NS, FPT5, and LDB are shown as bar charts in Figure 6.4.

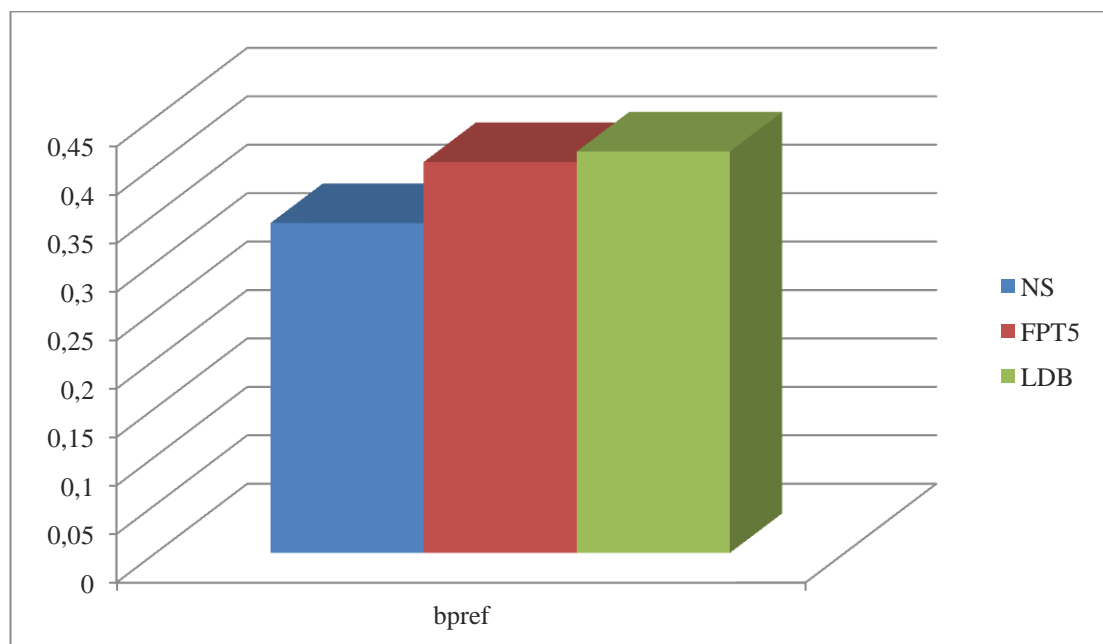


Figure 6.4 Visual representations of *bpref* values

The *bpref* or “binary preference” measure was introduced by Buckley and Voorhees (2004) which is designed for situations where relevance judgments are

incomplete. *Bpref* computes a preference relation of whether judged relevant documents are retrieved ahead of judged irrelevant documents (ignores the documents not evaluated by users). *Bpref* and *MAP* are very highly correlated when used with complete judgments. But when judgments are incomplete like the ones we use, rankings of systems by *bpref* still correlate highly to the original ranking, whereas rankings of systems by *MAP* do not.

In terms of *bpref*, LBD is better than the rest (2.68% better than FPT5, 21.67% better than NS). The *bpref* values of FPT5 and LBD are close to each other; on the other hand, P@10 and P@20 values of LBD are about 15% higher than that of FPT5.

6.3 Evaluation of Lemmatizer Performance

6.3.1 Parameters

We did a set of benchmarks on two different word sets, 100,000 words and 1,000,000 words; both are eliminated from stop words and constructed with random words taken from random documents of *Milliyet* dataset (created by Can & others., 2008).

Benchmarks were launched on the same environment which has the following configuration:

- Windows 7 64-bit OS
- Intel T9600 2.8 GHz CPU
- 4 GB RAM
- NVIDIA GT240M
- Cuda SDK version 3.2

Before we started to run tests, we selected “threads per block” parameter as 256 and defined “thread block count” parameter as “word count” / “threads per block” for all kernels. Also all algorithms tested have full warp occupancy.

6.3.2 Methods

We subjected the one sequential CPU-bound as a reference and its eight CUDA equivalent algorithms for benchmarking. Each test was run with both 100,000 and 1,000,000 words. The tests were conducted each utilizes different structures which are discussed at Section 5.1.1 for evaluation:

1. **LW:** Uses sequential algorithm with tree struct.(acronym of **L**emmatize **W**ord)
2. **LWAOs:** Uses parallel array of structs algorithm
3. **LWArrays:** Uses parallel arrays algorithm
4. **LWSOA:** Uses parallel struct of arrays algorithm
5. **LWCompactAOs:** Uses parallel array of structs algorithm with compact nodes
6. **LWCompactArrays:** Uses parallel arrays algorithm with compact nodes
7. **LWCompactSOA:** Uses parallel struct of arrays algorithm with compact nodes
8. **LWCompactSOAShared:** Uses parallel struct of arrays algorithm with compact nodes placed via pre-order traversal basis and exploits shared memory
9. **LWCompactSOABFS:** Uses parallel struct of arrays algorithm with compact nodes placed via breadth – first traversal basis and exploits shared memory

6.3.3 Results

For our first test we prepared 100,000 words and ran each algorithm 10 times to be sure on accuracy of results and then wrote down the obtained average time to Table 6.5. Results of benchmarking all of methods, compared by memory bandwidth and time consuming are summarized in Table 6.5.

The performance of each method to process 100,000 words is described in Table 6.5, where “Total Runtime” represents the time that it takes to copy the data to the

graphics card, call and execute the kernel, and copy the results from the graphics card back to system memory in milliseconds. "Bandwidth" column is rate at which data can be read from or stored into a memory. Memory bandwidth is usually expressed in units of bytes/second, "Total Speed up factor" column is value of CPU-bound kernel time divided by value of current kernel time.

Table 6.5 Results for 100,000 words

Algorithm	Type	Structure	Total Runtime in milliseconds	Bandwidth (GBps)	Total Speed up factor
LW	SEQ	Trie	2876.633	N/A	-
LWAOS	CUDA	AOS	58.809	11.41	48.92
LWArrays	CUDA	Arrays	58.232	11.53	49.40
LWSOA	CUDA	SOA	46.126	10.97	62.36
LWCompactAOS	CUDA	AOS	53.519	11.52	53.75
LWCompactArrays	CUDA	Arrays	57.787	11.66	49.78
LWCompactSOA	CUDA	SOA	39.075	13.18	73.62
LWCompactSOAShared	CUDA	SOA + Shared Memory	34.344	13.95	83.76
LWCompactSOABFS	CUDA	SOA via BFS + Shared Memory	32.284	14.84	89.10

The table clearly shows that the parallel algorithms outperform the sequential implementation. The speedup values of over 48x to 90x testify sufficient efficiency of our solution.

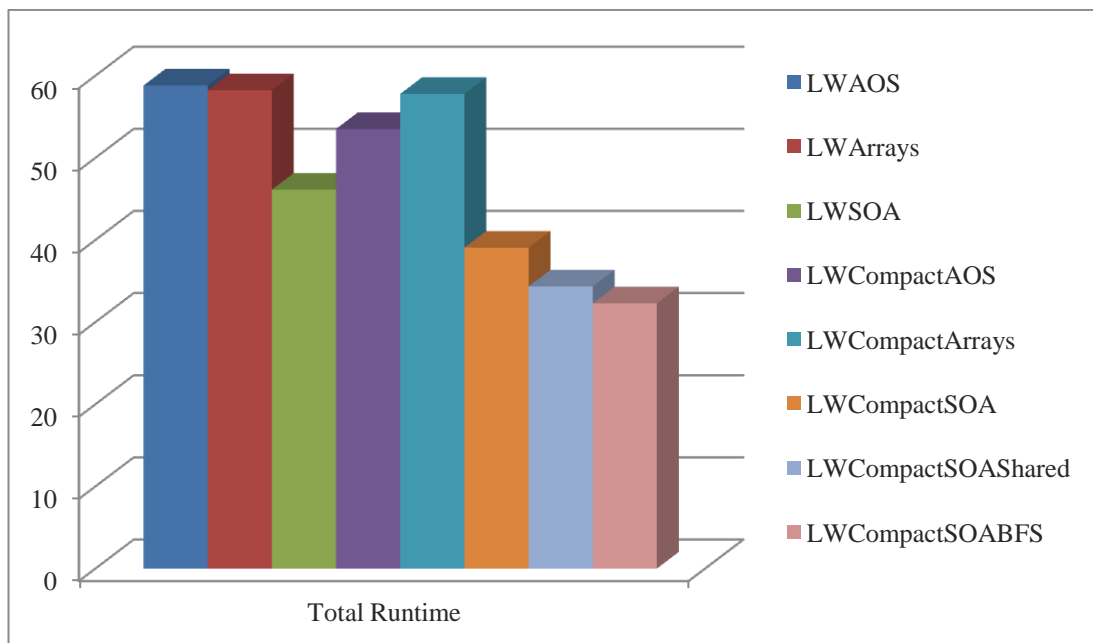


Figure 6.5 Visual representation of results for 100,000 words in terms of search runtime

Here, in Figure 6.5, the bar chart shows the total performance times for our eight lemmatizing algorithms applied on 100,000 words. Side by side, these bars show how performance is affected by structure selection and memory layout. Worth noticing is the performance of *LWCompactSOABFS* implementation is the best performing.

Later, we tested all algorithms on 1 million words set with the same methods applied in previous tests; in order to test effects of input data size on performance. And the results are as follows:

Table 6.6 Results for 1 million words

Algorithm	Type	Structure	Total runtime in milliseconds	Bandwidth (GBps)	Total Speed up factor
LW	SEQ	AOS	29076.238	N/A	-
LWAOS	CUDA	AOS	594.041	11.66	48.95
LWArrays	CUDA	Arrays	598.227	11.39	48.60
LWSOA	CUDA	SOA	475.802	10.83	61.11
LWCompactAOS	CUDA	AOS	590.683	11.41	49.23
LWCompactArrays	CUDA	Arrays	595.149	11.63	48.86
LWCompactSOA	CUDA	SOA	391.942	13.41	74.19
LWCompactSOAShared	CUDA	SOA+ Shared Memory	345.192	14.15	84.23
LWCompactSOABFS	CUDA	SOA via BFS + Shared Memory	324.315	14.86	89.65

From the Table 6.6, we can see that there is no significant difference from the results seen before in Table 6.5 Our data parallel algorithms outpace the sequential implementation with enormous speed up factors.

Also our GPU bandwidth performance (memory throughput) ratio to the GPU's (NVIDIA GeForce GT240M) theoretical bandwidth (25.6GBps) is good which is $14.86\text{GBps} / 25.6\text{GBps} = 58\%$ (GPU Specifications are added to Appendix 2). Ratio must be over 50% in order to be called good and 70% is very good (NVIDIA Corporation, August 2010).

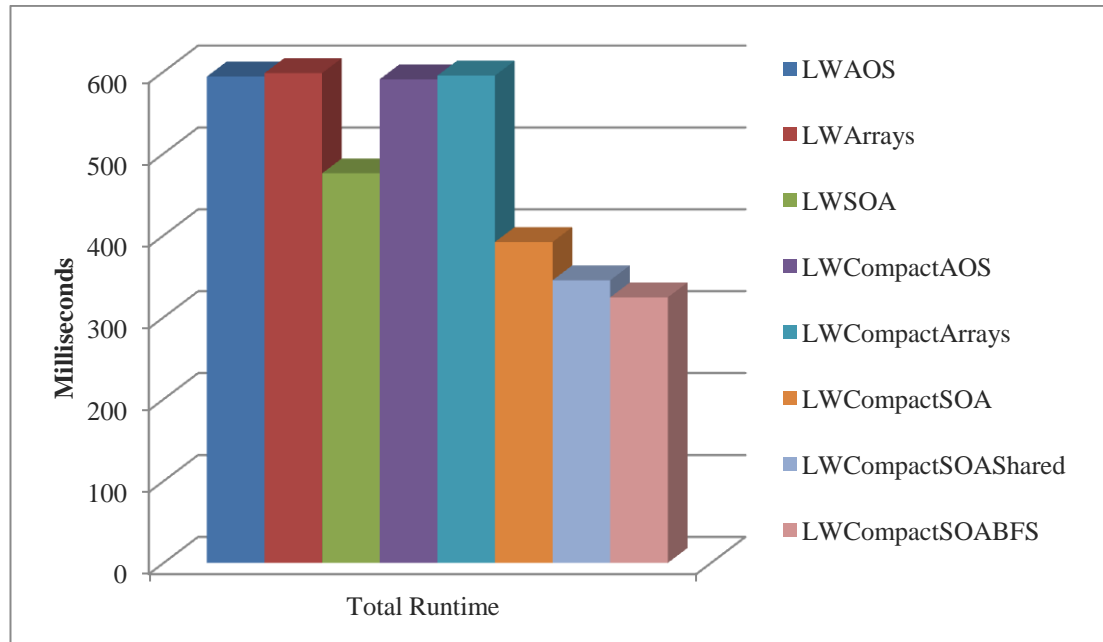


Figure 6.6 Visual representations of results for 1 million words

Apparently, *LWCompactSOABFS* implementation performance has turned out to be outperforming the other implementations again. This algorithm maps naturally to the GPU, exploiting its parallelism and cache, and this is reflected in the considerable speed increase over a CPU version by around 90 orders of magnitude.

Another performance increase can be observed in accessing the data located in GPU's memory, which is accelerated using the shared memory. However, it seems that the optimization using shared memory is significant, if we compare the performance gain between *LWCompactSOA* and *LWCompactSOAShared* algorithms which are sharing same structure but latter exploits shared memory; the timing measurements show that using the shared memory optimizes the execution time by at almost 15%.

So in brief, compared to the CPU baseline implementation, we achieved significant speed-up factors of the CUDA kernels to the sequential kernel ranging from 49x to 90x in our evaluations.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

In this thesis, we presented a hardware accelerated implementation of Turkish lemmatizing algorithm exploiting GPU devices through NVIDIA's CUDA and evaluation of it. The work enables researchers to easily utilize their CUDA device for lemmatizing big chunks of words from within in C or C++ applications. Our lemmatizer is based on CUDA, so, other cards supporting CUDA can also be used, and our approach can be ported to other programming environments.

Our study conclusively shows that lemmatizing is essential in the implementation of Turkish information retrieval systems. In our IR experiments, the most effective stemming method was our lemmatizer. The performance comparison of fixed prefix truncation (FPT5), our lemmatizer (LDB), and no stemming approaches (NS), shows that our lemmatizer performs better than the other two in terms of all measurements. The stemming option LDB provides 28.54% in terms of MAP; 79.70% in terms of GMAP; 16.92% in terms of P10; 18.58% in terms of P20 and 2.68% in terms of bpref, respectively, higher performance than that of FPT5.

But in terms of bpref, the measurements also show that FPT5 and LDB provide comparable results (with 2.68% difference), similar to work of Can & others. (2008) that showed for Turkish lemmatizer and a simple stemmer provide retrieval environments with similar bpref performances.

To streamline the overall results, it is clear that LDB produced the best results against other approaches in terms of all measurements. The FPT5 is also effective, but not as effective as LDB.

Even though floating-point calculations are not dominating our lemmatizing algorithm and its word processing characteristics limits the effectiveness due to non-synchronized branching and diverging, data dependent loop bounds, we achieved a significant speedup over the baseline algorithm on a CPU. More specifically, we

achieve up to a 90x speedup over CPU based sequential algorithm for the problem solution on selected word sets. This work demonstrates the potential of GPUs to accelerate even branch dominated massive word lemmatizing algorithms by carefully selecting and redesigning data structures and selecting appropriate memory types on hardware.

We took eight different approaches to the selection of an efficient data structure for CUDA programming model. We used our lemmatizer to lemmatize several different word sets and evaluated the performance of the eight parallel algorithms in comparison to a baseline implementation running on a single CPU. Our results showed that the parallel algorithms run significantly faster. More specifically our fastest algorithm (*LWCompactSOABFS*) achieved a speedup of around 90x in comparison to the baseline to perform lemmatization on a word set containing 1 million words. *LWCompactSOABFS* performs very well compared to the other algorithms, since its layout is compatible with the SIMD computation model of GPUs. The results confirm that; the struct of arrays implementation constructed with breadth-first traversal from trie offers best results for our lemmatizer.

Previous works on agglutinative languages (Can & others., 2008; Kettunen, Kunttu, & Jarvelin, 2005) show that lemmatizers are more effective than simple fixed prefix truncate but latter is preferable because the way its low complexity and simplicity. On the other hand , in this thesis, we show that we can exploit the gains of lemmatization via GPGPU, which provides us a more effective and efficient lemmatizer.

For future work there are several additional evaluations and improvements that are of interest. The algorithms we used for our evaluation all have similar characteristics. We may add some feedback mechanism to look up algorithm in order to increase accuracy. Feedback mechanism should allow look up procedure to turn back to parent node in case of the character looked up in trie is available for both transformation and key match (discussed in Section 2.3.4) or is available for two different haplology cases (node has both i and u narrow vowels and procedure must

choose one of them). Thus, if the procedure chooses wrong path it can turn back to parent node with the aim of proceeding through second route. But this will slow up the procedure a little bit and also will increase data dependency which is the most significant case to be avoided in parallel computing. So, in order to add this mechanism may be obliged to change the whole structure of procedure. Furthermore, our lemmatizer is working on only single words the lemmatizing process can be improved to handle phrases. Also the lemmatizer returns words meanings from dictionary which makes it perfect sub-tool for word sense disambiguation (WSD) programs. With our lemmatizer and a parallel WSD algorithm to select most appropriate lemma may result to a higher accuracy.

Also as the device memory, registers and shared memory increase, additional amounts of data can be processed in parallel. It is expected that future versions of CUDA and future NVIDIA devices will offer increased performance. To take advantage of performance increases with these developments, the structure of algorithm can be changed and variables can be placed on faster memories as an additional effort.

REFERENCES

- Altingovde, I.S., Ozcan, R., Ocalan, H.C., Can, F., Ulusoy, Ö. (2007). Large-scale cluster-based retrieval experiments on Turkish texts. *In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (ACM SIGIR '07)*, 891-892.
- Buckley, C., & Voorhees, E.M. (2004). Retrieval evaluation with incomplete information. *In Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval (ACM SIGIR '04)*, 25–32.
- Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval* (1st ed). NY: Addison-Wesley Professional.
- Can, F., Kocberber S., Balcik, E., Kaynak, C., Ocalan, H.C., Vursavas O.M. (2008). Information retrieval on Turkish texts. *JASIST*, 59 (3), 407-421.
- Can, F., Kocberber S., Balcik, E., Kaynak, C., Ocalan, H.C., Vursavas O.M. (2006). First large-scale information retrieval experiments on turkish texts. *In Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval (ACM SIGIR '06)*, 627 – 628.
- Frakes, W.B., & Baeza-Yates, R. (1992). *Information retrieval: algorithms and data structures*. Englewood Cliffs, NJ: Prentice Hall.
- Glaskowsky P. N. (September, 2009). *NVIDIA's Fermi: The First Complete GPU Computing Architecture*. Retrieved January 20, 2011, from http://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA's_Fermi-The_First_Complete_GPU_Architecture.pdf.

- Korenius, T., Laurikkala, J., Jarvelin, K., & Juhola, M. (2004). Stemming and lemmatization in the clustering of finnish text documents. *Proceedings of the 13th ACM International Conference on Information and Knowledge Management (ACM CIKM '04)*, 625 - 633
- Kettunen, K., Kunttu, T., & Jarvelin, K. (2005). To stem or lemmatize a highly inflectional language in a probabilistic IR environment?. *Journal of Documentation*, 61 (4), 476–496.
- Kirk, D.B., & Hwu, W.W. (2011). *Programming massively parallel processors a hands-on approach*. Burlington: Morgan Kaufmann.
- Manavski, S.A., Valle G. (2008). CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics*, 9 (2), 10+.
- Manning, C.D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge: Cambridge University Press.
- Morrison, D.R. (1968). Practical Algorithm to Retrieve Information Coded in Alphanumeric. *JACM*, 15 (4), 514 – 534.
- NVIDIA Corporation. (2009). *NVIDIA's next generation CUDA compute architecture Fermi v1.1*. Retrieved June 09, 2011, from http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.
- NVIDIA Corporation. (November, 2010). *NVIDIA CUDA programming guide version 3.2*. Retrieved November 22, 2010, from http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf.

- NVIDIA Corporation. (August, 2010). *NVIDIA CUDA C best practices guide version 3.2*. Retrieved November 22, 2010, from http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Best_Practices_Guide.pdf.
- NVIDIA Corporation. (October, 2010). *Compute visual profiler user guide*. Retrieved November 22, 2010, from http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/VisualProfiler/Compute_Visual_Profiler_User_Guide.pdf.
- Nvidia Corporation (June, 2009). *Specifications of Geforce GT240M GPU*. Retrieved August 13, 2011 from http://www.nvidia.com/object/product_geforce_gt_240m_us.html.
- Sanders, J., & Kandrot, E. (2011). *CUDA by example : an introduction to general-purpose GPU programming*. Boston: Addison-Wesley.
- Sanderson, M., & Zobel, J. (2005). Information retrieval system evaluation: Effort, sensitivity, and reliability. *In Proceedings of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval (ACM SIGIR '05)*, 162–169.
- Schatz, M.C., Trapnell, C., Delcher, A.L., Varshney, A. (2007). High-throughput sequence alignment using Graphics Processing Units. *BMC Bioinformatics*, 8 (1), 474+.
- Solak, A., & Can, F. (1994). Effects of stemming on Turkish text retrieval. *In Proceedings of the 9th International Symposium on Computer and Information Sciences*, 49–56.
- Shimpi A.L., & Wilson D. (June 16, 2008). *NVIDIA's 1.4 Billion Transistor GPU: GT200 Arrives as the GeForce GTX 280 & 260*. Retrieved December 16, 2010 from <http://www.anandtech.com/show/2549/2>.

Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E.P. & Ioannidis, S., (2008). Gnort: High Performance Network Intrusion Detection Using Graphics Processors. *Lecture Notes in Computer Science*, 5230, 116 – 134.

APPENDIX 1

Table A. 1 Evaluation measures of trec_eval program

Name	Description
num_ret	Total number of documents retrieved over all queries
num_rel	Total number of relevant documents over all queries
num_rel_ret	Total number of relevant documents retrieved over all queries
map	Mean Average Precision (MAP)
gm_map	Average Precision. Geometric Mean, q_score=log(MAX(map,.00001))
Rprec	R-Precision (Precision after R (= num-rel for topic) documents retrieved)
bpref	Binary Preference, top R judged nonrel
recip_rank	Reciprocal rank of top relevant document
iprec@recall_0.00	Interpolated Recall - Precision Averages at 0.00 recall
iprec@recall_0.10	Interpolated Recall - Precision Averages at 0.10 recall
iprec@recall_0.20	Interpolated Recall - Precision Averages at 0.20 recall
iprec@recall_0.30	Interpolated Recall - Precision Averages at 0.30 recall
iprec@recall_0.40	Interpolated Recall - Precision Averages at 0.40 recall
iprec@recall_0.50	Interpolated Recall - Precision Averages at 0.50 recall
iprec@recall_0.60	Interpolated Recall - Precision Averages at 0.60 recall
iprec@recall_0.70	Interpolated Recall - Precision Averages at 0.70 recall
iprec@recall_0.80	Interpolated Recall - Precision Averages at 0.80 recall
iprec@recall_0.90	Interpolated Recall - Precision Averages at 0.90 recall
iprec@recall_1.00	Interpolated Recall - Precision Averages at 1.00 recall
P@5	Precision after 5 docs retrieved
P@10	Precision after 10 docs retrieved
P@15	Precision after 15 docs retrieved
P@20	Precision after 20 docs retrieved
P@30	Precision after 30 docs retrieved
P@100	Precision after 100 docs retrieved
P@200	Precision after 200 docs retrieved
P@500	Precision after 500 docs retrieved
P@1000	Precision after 1000 docs retrieved

APPENDIX 2

Table A. 2 Specifications of NVIDIA GeForce GT240M GPU (NVIDIA Corporation, June 2009)

Model		GeForce GT 240M
Year		June 15, 2009
Code name		GT216
Fab (nm)		40
Businterface		PCIe 2.0 x16
Memory max (MiB)		1024
Clock speed	Core (MHz)	550
	Shader (MHz)	1210
	Memory (MHz)	1600
Config core		48:16:08
Memory	Bandwidth max (GB/s)	25.6
	Bus type	GDDR3
	Bus width (bit)	128
Graphics library support (version)	DirectX	10.1
	OpenGL	3.3
GFLOPs (MADD/MUL)		174
TDP (Watts)		23

APPENDIX 3

Table A. 3 Stop words list

Stop Words					
a	bunu	hem	mu	s	vb
acaba	bunun	henüz	mü	sana	var
altı	burada	hep	n	sekiz	veya
ama	bütün	hepsi	nasıl	sen	veyahut
ancak	c	hepsine	ne	senden	y
artık	ç	hepsini	ne kadar	seni	ya
asla	çoğu	her	ne zaman	senin	ya da
aslında	çoğuna	her biri	neden	siz	yani
az	çoğunu	herkes	nedir	sizden	yedi
b	çok	herkese	neler	size	yerine
bana	çünkü	herkesi	nerde	sizi	yine
bazen	d	hiç	nerede	sizin	yoksa
bazı	da	hiç kimse	nereden	sonra	z
bazıları	daha	hiçbiri	nereye	ş	zaten
bazısı	de	hiçbirine	nesi	şayet	zira
belki	değil	hiçbirini	neyse	şey	
ben	demek	ı	niçin	şeyden	
beni	diğer	i	niye	şeye	
benim	diğeri	için	o	şeyi	
beş	diğerleri	içinde	on	şeyler	
bile	diye	iki	ona	şimdi	
bir	dokuz	ile	ondan	şöyle	
birçoğu	dolayı	ise	onlar	şu	
birçok	dört	işte	onlara	şuna	
birçokları	e	j	onlardan	şunda	
biri	elbette	k	onların	şundan	
birisi	en	kaç	onların	şunlar	
birkaç	f	kadar	onu	şunu	
birkaçı	fakat	kendi	onun	şunun	
birşey	falan	kendine	orada	t	
birşeyi	felan	kendini	oysa	tabi	
biz	filan	ki	oysaki	tamam	
bize	g	kim	ö	tüm	
bizi	gene	kime	öbürü	tümü	
bizim	gibi	kimi	ön	u	
böyle	ğ	kimin	önce	ü	
böylece	h	kimisi	ötürü	üç	
bu	hâlâ	l	öyle	üzere	
buna	hangisi	m	p	üzerine	
bunda	hani	madem	r	v	
bundan	hatta	mı	rağmen	ve	

APPENDIX 4

Table A. 4 Our stop words list. (We applied fixed prefix truncate by 5 to words on Appendix 3)

Stop Words				
a	ç	ı	on	şuna
acaba	çoğu	ı	ona	şunda
altı	çoğun	için	ondan	şunla
ama	çok	iki	onlar	şunu
ancak	çünkü	ile	onu	şunun
artık	d	ise	onun	t
asla	da	işte	orada	tabi
aslın	daha	j	oysa	tamam
az	de	k	oysak	tüm
b	değil	kaç	ö	tümü
bana	demek	kadar	öbürü	u
bazen	diğer	kendi	ön	ü
bazı	diye	ki	önce	üç
bazıl	dokuz	kim	ötürü	üzere
bazıs	dolay	kime	öyle	üzeri
belki	dört	kimi	p	v
ben	e	kimin	r	ve
beni	elbet	kimis	rağme	vb
benim	en	l	s	var
beş	f	m	sana	veya
bile	fakat	madem	sekiz	veyah
bir	falan	mı	sen	y
birço	felan	mi	sende	ya
biri	filan	midir	seni	yani
biris	g	mu	senin	yedi
birka	gene	mudur	siz	yerin
birşe	gibi	mü	sizden	yine
biz	ğ	n	size	yoksa
bize	h	nasıl	sizi	z
bizi	hâlâ	ne	sizin	zaten
bizim	hangi	neden	sonra	zira
böyle	hani	nedir	ş	
bu	hatta	neler	şayet	
buna	hem	nerde	şey	
bunda	henüz	nered	şeyde	
bunla	hep	nerey	şeye	
bunu	hepsi	nesi	şeyi	
bunun	her	neyse	şeyle	
burad	herke	niçin	şimdi	
bütün	hiç	niye	şöyle	
c	hiçbi	o	şu	

APPENDIX 5

Table A. 5 Queries in the dataset. We used the queries in the “description” column.

QueryID	Topic	Description
235	Kuş Gribi	Kuş gribi nedir, nasıl bulaşır, belirtileri nelerdir sorularına cevap olabilecek dokümanlar.
238	Kıbrıs Sorunu	Türkiye'nin Avrupa Birliği'ne tam üyelik sürecinde Kıbrıs sorununu ele alan bir doküman.
241	Üniversiteye giriş sınavı	Türkiye'de üniversiteye giriş sınavının gençler üzerindeki etkileri, gençlerin ve kamuoyunun bu sınav için düşündükleri.
243	Tsunami	Güney Asya'yı 26 Aralık 2004'te vuran büyük Tsunami faciası ve bu faciannın sonuçları.
244	Mavi Akım Doğalgaz Projesi	Mavi akımın ulusal enerji politikamızdaki yeri, ekonomik maliyeti
258	Deprem Tedbir Önlem	Büyük bir bölümü deprem bölgesi olan Türkiye'de deprem öncesi alınan tedbirler nelerdir
265	Türkiye PKK çatışmaları	Türk Silahlı Kuvvetleri ile PKK arasında meydana gelen çatışmalar
270	Film Festivalleri	Türkiye' de gerçekleştirilen film festivalleri ve bu festivallerde ödül alan sanatçılar.
271	Bedelli askerlik uygulaması	Askerlik hizmetinin bedelli olarak yapılmasının Türk kamuoyu üzerindeki etkileri, ilgili makamların söz konusu uygulama hakkındaki görüşleri.
278	Stresle Başa Çıkma Yolları	Günlük hayatımızı birçok yönden olumsuz etkileyen stresle nasıl mücadele edebiliriz
282	Şampiyonlar Ligi	Futbol Avrupa Şampiyonlar ligi 2004-2005 sezonu mücadelesi
283	17 Ağustos Depremi	17 Ağustos Depreminin Türkiye üzerindeki sosyal ve ekonomik etkileri
284	Türkiye'de internet kullanımı	Son yıllarda bilişim teknolojisinin gelişmesiyle internet kullanımının artması, kullanıcı profili, kullanım amaçları.
288	Amerika Irak işgal demokrasi petrol	Amerika'nın Irak operasyonu demokrasi adına yapılmış bir hareket midir yoksa petrol için yapılan bir işgal midir?
289	Türkiye'de futbol şikesi	Şikenin Türk futbolundaki yeri, etkisi, yarattığı sonuçlar, bu konuda alınan tedbirler, verilen cezalar, uzman görüşleri.
294	Fadıl Akgündüz	Fadıl Akgündüz'ün milletvekili olamayacağına ilişkin

		yapılan itirazlar.
295	İşsizlik sorunu	Türkiye'de işsizlik sorununun bireylerin ruhsal sağlığı üzerindeki olumsuz etkileri, işsizliğin toplumsal ve ekonomik sonuçları.
296	2005 F1 Türkiye Grand Prix	Formula 1'de 2005 sezonun 14'üncü yarışı Türkiye Grand Prix'sini rakamlarla anlatan bir doküman.
298	Ekonomik kriz	Türkiye'de ekonomik krize neden olan olaylar.
300	Nuri Bilge Ceylan	Nuri Bilge Ceylan sinemasının Türk sinemasına etkileri
301	Türkiye'de meydana gelen depremler	Türkiye'de meydana gelen depremlerin insanlar üzerindeki etkileri ve bu depremlere karşı alınan önlemler.
302	ABD-İrak Savaşı	ABD ve İngiltere'nin Irak'a yönelik başlattığı saldırının ardından tarafların kayıplarını açıklayan bir doküman.
304	Hakan Şükür'ün milli takım kadrosuna alınmaması	Ersun Yanal Hakan Şükür'ü A milli futbol takımı kadrosuna dahil etmeme kararı doğru mu yanlış mı Ersun Yanal haklı mı haksız mı
305	Avrupa Birliği, Türkiye ve insan hakları	Türkiye'nin Avrupa Birliği'ne (AB) uyum sürecinde insan haklarıyla ilgili yaptığı yenilikler, çıkardığı kanunlar
306	Turizm	Son yıllarda Türk turizmindeki gelişmeler
307	Türkiye'deki sokak çocukları	Türkiye'deki özellikle İstanbul'daki sokak çocuklarıyla ilgili olarak yapılan çalışmalar, bu çocukların sokak çocuğu olma nedenleri, parçalanmış ailelerin bu olaya etkileri, bu çocukların sayıları, olayın toplumsal etkileri, bu çocukların işlediği suçlar.
308	Türk filmleri ve sineması	Son yıllarda büyük sıçrama yaptığı söylenen Türk sinemasında yeni parlayan isimler, en kayda değer filmler, eski ustaların bu konudaki katkıları.
311	Pakistan Depremi	Pakistan'da 8 Ekim'de meydana gelen büyük deprem ve bu depremin sonuçları
324	Sanat ödülleri	Türkiye'de edebiyat, müzik, resim, sinema gibi sanat dallarında verilmiş ödüller.
339	Avrupa Birliği Fonları	Avrupa Birliği tarafından Türkiye'de, kamuya ve özel sektöre ait her alandaki proje ve programlar için ayrılan fonlar, geri ödemeli veya hibeli krediler.
342	Futbolda şike	Futbolda şike söylentileri, yorumlar ve kanıtlar
343	milletvekili dokunulmazlığı	Milletvekilleri meclis kararı olmadan yargılanamaz, soruşturmaya tabii tutulamaz.
344	2001 Erkekler Avrupa	milli takımı sporcularının turnuva süresindeki ve turnuva

	Basketbol Şampiyonası	sonrasındaki düşünceleri, onlarla yapılan röportajlar ve takımdaki son haberler
347	2002 Dünya Kupası	Türk Milli Takımı'nın 3. olduğu 2002 Dünya Kupası
348	bilişim eğitimi ve projeleri	Türkiye'de yapılan bilişim eğitimi ve bilişim projeleri, bu eğitimin ve projelerin kaliteleri ve sanayiye katkıları
349	Global ısınma	Global ısınmanın dünya iklimine olumsuz etkileri nelerdir, bu etkileri azaltmak veya yok etmek için neler yapmalıyız?
350	Türkiye'de mortgage	Mortgage'in nasıl işleyeceği, Türkiye'ye yararları ve mevcut kredi sistemleri üzerindeki oluşturacağı etki. Kamuoyunun mortgage'den beklentileri.
352	ABD Afganistan Savaşı	ABD'nin Afganistan'a yaptığı operasyonda Türkiye'nin rolünü açıklayan bir doküman.
360	Yüzüklerin Efendisi-Kralın Dönüşü	11 dalda ödül alan Yüzüklerin Efendisi-Kralın Dönüşü filminin başarısını anlatan bir doküman.
362	Beyin Göçü	Türkiye'de yetişen akademik olarak başarılı öğrencilerin üniversite veya sonrasındaki bilimsel çalışmaları için yurt dışını tercih etmeleri
366	aile kadın şiddet	Aile içinde kadına karşı uygulanan şiddetin alkol ve parasızlık gibi sebepler dışında ne gibi sebepleri vardır Kadına şiddet daha çok hangi tür toplumlarda görülmektedir Çocuk gelişimine etkileri nelerdir
367	sporcuların doping yapması	Sporcuların doping yapması yarışma veya müsabakalarda fiziksel dayanıklılıklarını artırmak için kullanımı yasak olan performans artırıcı maddeleri kullanmasıdır.
368	ozon tabakasındaki delik	Ozon tabakası dünyaya uzaydan gelen ultraviyole ışınları süzen bir filtredir. Bu filtrede oluşan delik cilt kanseri vakalarında artışa neden olmaktadır.
373	Rusya'da okul baskını	Kuzey Osetya'da yüzlerce kişinin rehin tutulduğu okul binasına Rus güçleri tarafından düzenlenen operasyon.
374	İstanbul'da bombalı saldırı	İstanbul'da 15 Kasım 2003 tarihinde, Kuledibi'ndeki Neve Şalom ve Şişli'deki Betyaakov Sinagogu yakınlarında saat 09.30'da meydana gelen patlamalar.
377	Sakıp Sabancı'nın vefatı	Sakıp Sabancı'nın 10 Nisan 2004 saat 05.55 sıralarında vefat etmesiyle ilgili dokümanlar.
378	Ecevit Sezer çatışması	MGK toplantısında Cumhurbaşkanı Sezer'in Başbakan Ecevit'e anayasayı fırlatmasıyla gelişen olaylar.

382	Kıbrıs Türk üniversiteleri	Kıbrıs'ta açılan yeni üniversitelerin ve burada okuyan öğrencilerin sorunları, nasıl öğrenci aldıkları, denklik, kalacak yurt, öğretim üyesi bulma konusunda yaşanan sorunlar.
383	Türkiye'de 2003 yılında turizm	Türkiye'ye 2003'te gelen turist sayısı ve dağılımı, illerdeki turizm durumu, turizmin ekonomiye katkısı,
406	Türkiye'nin Nükleer santral çalışmaları	Türkiye'nin Nükleer santral çalışmaları, nükleer santral projeleri
411	hızlı tren kazası	hızlı tren kazasının nedenleri ve alınan önlemler
412	YÖK'ün Üniversitelerimiz üzerindeki etkisi	Yüksek Öğretim Kurulu, YÖK'ün kuruluşu, üniversitelerimiz üzerindeki olumlu olumsuz etkileri, eleştirilen yönleri, YÖK hükümet ilişkileri
414	İbrahim Tatlıses'in kadınları	İbrahim Tatlıses'in yaşadığı aşklar ve kadınlarla ilgili yarattığı huzursuzluklar kavgalar.
416	Parçalanmış aileler	Parçalanmış aile bireylerinin yaşadığı sorunlar, özellikle bu türden ailelerin çocuklarının ve kadınlarının durumları.
417	Aile içi şiddet	Aile bireyleri arasında yaşanan şiddet olayları ve sebepleri. Çocuklara ve kadınlara uygulanan şiddet, buna maruz kalanların yaşadığı sorunlar.
419	Türkiye'de kanser	Türkiye'de son yıllarda özellikle Karadeniz bölgesinde arttığı düşünülen kanserli hasta sayısının Çernobil olayı ile varsa olan ilişkisi ve bu ilişkiyi irdeleyen çalışmalar, resmi kuruluşlar tarafından verilen istatistiklerin güvenilirliği.
421	Futbol terörü ve holiganizm	Futbolda yaşanan şiddet olayları, bunların nedenleri ve engellenmesi için alınacak önlemler.
423	Türkiye'de ikinci el otomobil piyasası	Türkiye'de son yıllarda ikinci el otomobil piyasasındaki durum, son dönemlerde piyasada yaşanan düşüşün sebep ve sonuçları, yeni otomobil piyasasındaki yeniliklerle bağlantısı
424	Tarihi eser kaçakçılığı	Türkiye'den kaçırılan tarihi eserler ve tarihi eser kaçakçılığa karşı yapılanlar
426	Festival	İnsanların eğlenmesi ve kültür paylaşımı yapabilmesi için düzenlenen festivaller.
428	Türkiye'de bayram tatillerinde meydana gelen trafik kazaları	Türkiye'de bayram tatillerinde meydana gelen trafik kazalarının nedenleri, ve alınan önlemler.
432	öğrenmeyi etkileyen faktörler	öğrenmeyi etkileyen faktörler ve etkileri, öğrenme teknikleri

433	Kekik otu	Kekik otunun faydaları, sağlık üzerindeki etkileri
435	teelif hakları	Türkiye'de telif hakkı yasalarının durumu ve bu konuda yapılan çalışmalar
437	İnternet ve toplum	İnternet'in yaygınlaşması, sunulan hizmetler, toplum üzerindeki etkileri.
442	Tarım Hayvancılık Sorunları	Türkiye'de tarım ve hayvancılık alanında yaşanan problemler ve bunların çözüm yolları.
444	İran'da Nükleer Enerji	İran'ın nükleer enerji ile ilgili politikaları, açıklamaları, nükleer enerji ile ilgili İran'da sürdürülen faaliyetler, uluslararası toplumdan İran'a nükleer enerji politikaları ile ilgili yöneltilen tepkiler veya verilen destekler
450	satranç	Satrancın yazılı basında ne ölçüde yer aldığı
452	Kalıtsal Hastalıklar	Genlerin insan sağlığı üzerindeki etkisi, hastalıkların kalıtsal nedenleri.
472	hiperaktivite ve dikkat eksikliği	hiperaktivite ve dikkat eksikliği nedir Belirtileri, teşhisi, tedavisi nelerdir Çocukların ve yetişkinlerin günlük yaşamına olumlu ve olumsuz etkileri nedir Hiperaktif çocuklara öğretmen nasıl yaklaşmalı Bu çocuklara yönelik eğitim sistemi nasıl geliştirilebilir
474	lenf kanseri	Türkiye'deki lenf kanser istatistikleri
481	28 Şubat süreci	28 Şubat süreci ve Türkiye üzerindeki etkileri

APPENDIX 6

Table A. 6 Lemmatized queries

Query	Lemmatized Query	Accuracy	Summary
Kuş gribi nedir, nasıl bulaşır, belirtileri nelerdir sorularına cevap olabilecek dokümanlar.	kuş grip bulaş belirti soru cevap olabil doküman	100%	
Türkiye'nin Avrupa Birliği'ne tam üyelik sürecinde Kıbrıs sorununu ele alan bir doküman.	türkiye avrupa birlik tam üyelik süreç kıbrıs sorun ele alan doküman	90.91%	The lemma of “ele” must be “el” (hand) but our lemmatizer returns “ele”+“(mek)” (to eliminate) And the lemma of “alan” must be “al”+“(mak)” (to take) But our lemmatizer return “alan”(region)
Türkiye’de üniversiteye giriş sınavının gençler üzerindeki etkileri, gençlerin ve kamuoyunun bu sınav için düşündükleri.	türkiye üniversite giriş sınav genç etki genç kamuoyu sınav düşün	100%	
Güney Asya’yı 26 Aralık 2004’te vuran büyük Tsunami faciası ve bu facianın sonuçları.	güney asya 26 aralık 2004 vur büyük tsunami facia facia sonuç	100%	
Mavi akımın ulusal enerji politikamızdaki yeri, ekonomik maliyeti	mavi akım ulusal enerji politika yer ekonomik maliyet	100%	
Büyük bir bölümü deprem bölgesi olan Türkiye’de deprem öncesi alınan tedbirler nelerdir?	büyük bölüm deprem bölge ol türkiye deprem önce alın tedbir	100%	
Türk Silahlı Kuvvetleri ile PKK arasında meydana gelen çatışmalar	türk silahlı kuvvet pkk arası meydan gelen çatışma	100%	
Türkiye’ de gerçekleştirilen film festivalleri ve bu festivallerde ödül alan sanatçılar.	türkiye gerçekleştiril film festival festival ödül alan sanatçı	87.5%	The lemma of “alan” must be “al”+“(mak)” (to take) But our

			lemmatizer return “alan”(region)
Askerlik hizmetinin bedelli yapılmasının Türk kamuoyu üzerindeki etkileri, ilgili makamların söz konusu uygulama hakkındaki görüşleri.	askerlik hizmet bedelli yapılma türk kamuoyu etki ilgili makam söz konu uygulama hakkında görüş	100%	
Günlük hayatımızı birçok yönden olumsuz etkileyen stresle nasıl mücadele edebiliriz ?	günlük hayat yön olumsuz etkile stres mücadele ede	87.5%	The lemma of “ede” must be “et”+“(mek)” (to do) But our lemmatizer return “ede”(brother)
Futbol Avrupa Şampiyonlar ligi 2004-2005 sezonu mücadelesi	futbol avrupa şampiyon lig 2004-2005 sezon mücadele	100%	
17 Ağustos Depreminin Türkiye üzerindeki sosyal ve ekonomik etkileri	17 ağustos deprem türkiye sosyal ekonomik etki	100%	
Son yıllarda bilişim teknolojisinin gelişmesiyle internet kullanımının artması, kullanıcı profili, kullanım amaçları.	son yıl bilişim teknoloji gelişme internet kullanım artma kullanıcı profil kullanım amaç	100%	
Amerika'nın Irak operasyonu demokrasi adına yapılmış bir hareket midir yoksa petrol için yapılan bir işgal midir?	amerika irak operasyon demokrasi adına yapılmış hareket petrol yapılan işgal	90%	The lemma of “yapılan” must be “yapıl”+“(mak)” (be done) But our lemmatizer return “yapılan”+“(mak)” (to be settled)
Şikenin Türk futbolundaki yeri, etkisi, yarattığı sonuçlar, bu konuda alınan tedbirler, verilen cezalar, uzman görüşleri.	şike türk futbol yer etki yarat sonuç konu alın tedbir veril ceza uzman görüş	100%	
Fadıl Akgündüz'ün milletvekili olamayacağına ilişkin yapılan itirazlar.	fadıl akgündüz milletvekili ol ilişkin yapılan itiraz	85.71%	The lemma of “yapılan” must be “yapıl”+“(mak)” (be done) But our lemmatizer return “yapılan”+“(mak)” (to be settled)
Türkiye'de işsizlik sorununun	türkiye işsizlik sorun birey	100%	

bireylerin ruhsal sađlıđı üzerindeki olumsuz etkileri, işsizliđin toplumsal ve ekonomik sonuçları.	ruhsal sađlık olumsuz etki işsizlik toplumsal ekonomik sonuç		
Formula 1'de 2005 sezonun 14'üncü yarışı Türkiye Grand Prix'sini rakamlarla anlatan bir doküman.	form 1 2005 sezon 14 yarış türkiye grand pr rakam anlat doküman	91.67%	“Formula” and “Prix“ are not Turkish words and therefore they don't take place in Turkish dictionary. Lemmatizer returns latest found lemma.
Türkiye'de ekonomik krize neden olan olaylar.	türkiye ekonomik kriz ol olay	100%	
Nuri Bilge Ceylan sinemasının Türk sinemasına etkileri	nuri bilge ceylan sinema türk sinema etki	100%	
Türkiye'de meydana gelen depremlerin insanlar üzerindeki etkileri ve bu depremlere karşı alınan önlemler.	türkiye meydan gelen deprem insan etki deprem karşı alın önlem	100%	
ABD ve İngiltere'nin Irak'a yönelik başlattığı saldırının ardından tarafların kayıplarını açıklayan bir doküman.	abd ingiltere irak yönelik başlat saldırı ardı taraf kayıp açıkla doküman	100%	
Ersun Yanal Hakan Şükür'ü A millî futbol takımı kadrosuna dahil etmeme kararı doğru mu yanlış mı Ersun Yanal haklı mı haksız mı	ersun yanal hakan şükür millî futbol takım kadro dahil etme karar doğru yanlış ersun yanal haklı haksız	100%	
Türkiye'nin Avrupa Birliği'ne (AB) uyum sürecinde insan haklarıyla ilgili yaptığı yenilikler, çıkardığı kanunlar	türkiye avrupa birlik ab uyum süreç insan hak ilgili yap yenilik çıkar kanun	100%	
Son yıllarda Türk turizmindeki gelişmeler	son yıl türk turizm gelişme	100%	
Türkiye'deki özellikle İstanbul'daki sokak çocuklarıyla ilgili yapılan çalışmalar, bu çocukların sokak çocuđu olma nedenleri, parçalanmış ailelerin bu olaya etkileri, bu çocukların sayıları, olayın toplumsal	türkiye özellikle istanbul sokak çocuk ilgili yapılan çalışma çocuk sokak çocuk olma parçalan aile olay etki çocuk sayı olay toplumsal etki çocuk işlet suç	95.83%	The lemma of “yapılan” must be “yapıl”+“(mak)” (be done). But our lemmatizer return “yapılan”(mak) (to be settled)

etkileri, bu çocukların işlediği suçlar.			
Son yıllarda büyük sıçrama yaptığı söylenen Türk sinemasında yeni parlayan isimler, en kayda değer filmler, eski ustaların bu konudaki katkıları.	son yıl büyük sıçrama yap söylen türk sinema yeni parla isim kay değer film eski usta konu katkı	94.44%	The lemma of “kayda” must be “kayıt” (registration) But our lemmatizer returns “kay”+”(mak)” (to slide). This problem occurs because there are valid entries like “ kaydırmak ” and “ kaydetmek ” so lemmatizer goes to “kayd” on trie then can’t find any match and returns the latest lemma (“ kay”).
Pakistan’da 8 Ekim’de meydana gelen büyük deprem ve bu depremin sonuçları	pakistan 8 ekim meydan gelen büyük deprem deprem sonuç	100%	
Türkiye’de edebiyat, müzik, resim, sinema gibi sanat dallarında verilmiş ödüller.	türkiye edebiyat müzik resim sinema sanat dal veril ödül	100%	
Avrupa Birliği tarafından Türkiye’de, kamuya ve özel sektöre ait her alandaki proje ve programlar için ayrılan fonlar, geri ödemeli veya hibeli krediler.	avrupa birlik tarafından türkiye kamu özel sektör ait alan proje program ayrıl fon geri ödemeli hibe kredi	100%	
Futbolda şike söylentileri, yorumlar ve kanıtlar	futbol şike söylenti yorum kanıt	100%	
Milletvekili meclis kararı olmadan yargılanamaz, soruşturmaya tabii tutulamaz.	milletvekili meclis karar olma yargılan soruşturma tabii tutul	100%	
milli takımı sporcularının turnuva süresindeki ve turnuva sonrasındaki düşünceleri, onlarla yapılan röportajlar ve takımdaki son haberler	milli takım sporcu turnuva süresinde turnuva düşünce yapılan röportaj takım haber	100%	
Türk Milli Takımı’nın 3. olduğu	türk milli takım 3 ol 2002	100%	

2002 Dünya Kupası	dünya kupa		
Türkiye'de yapılan bilişim eğitimi ve bilişim projeleri, bu eğitimin ve projelerin kaliteleri ve sanayiye katkıları	türkiye yapılan bilişim eğitim bilişim proje eğitim proje kalite sanayi katkı	100%	
Global ısınmanın dünya iklimine olumsuz etkileri nelerdir, bu etkileri azaltmak veya yok et için neler yapmalıyız?	global ısınma dünya iklim olumsuz etki etki azalt yok et yapma	100%	
Mortgage'in nasıl işleyeceği, Türkiye'ye yararları ve mevcut kredi sistemleri üzerindeki oluşturacağı etki. Kamuoyunun mortgage'den beklentileri.	mor işle türkiye yarar mevcut kredi sistem oluştur etki kamuoyu mor beklenti	83.33%	“Mortgage” is not a Turkish word and doesn't take place in Turkish dictionary. Lemmatizer returns latest found lemma.
ABD'nin Afganistan'a yaptığı operasyonda Türkiye'nin rolünü açıklayan bir doküman.	abd afganistan yap operasyon türkiye rol açıkla doküman	100%	
11 dalda ödül alan Yüzüklerin Efendisi-Kralın Dönüşü filminin başarısını anlatan bir doküman.	11 dal ödül alan yüzük efendi dönüş film başarı anlat doküman	90.91%	The lemma of “alan” must be “al”+“(mak) (to take) But our lemmatizer returns “alan”(region)
Türkiye'de yetişen akademik başarılı öğrencilerin üniversite veya sonrasındaki bilimsel çalışmaları için yurt dışını tercih etmeleri	türkiye yetişen akademik başarılı öğrenci üniversite bilimsel çalışma yurt dış tercih etme	100%	
Aile içinde kadına karşı uygulanan şiddetin alkol ve parasızlık gibi sebepler dışında ne gibi sebepleri vardır ? Kadına şiddet daha çok hangi tür toplumlarda görülmektedir? Çocuk gelişimine etkileri nelerdir?	aile içinde kadın karşı uygulan şiddet alkol parasızlık sebep dışında sebep vardır kadın şiddet tür toplum görül çocuk gelişim etki	95%	The lemma of “alan” must be “var” (available) But our lemmatizer returns “vardır”+“(mak)” (to let a matter reach)
Sporcuların doping yapması yarışma veya müsabakalarda fiziksel dayanıklılıklarını artırmak için kullanımı yasak olan performans artırıcı maddeleri kullanmasıdır.	sporcu doping yapma yarışma müsabaka fiziksel dayanıklılık artır kullanım yasak ol performans artırıcı madde kullanma	100%	

Ozon tabakası dünyaya uzaydan gelen ultraviyole ışınları süzen bir filtredir. Bu filtrede oluşan delik cilt kanseri vakalarında artışa neden olmaktadır.	ozon tabaka dünya uzay gelen ultraviyole ışın süzen filtre filtre oluş delik cilt kanser vaka artış ol	100%	
Kuzey Osetya'da yüzlerce kişinin rehin tutulduğu okul binasına Rus güçleri tarafından düzenlenen operasyon.	kuzey osetya yüz kişi rehin tutul okul bina rus güç tarafından düzenlen operasyon	100%	
İstanbul'da 15 Kasım 2003 tarihinde, Kuledibi'ndeki Neve Şalom ve Şişli'deki Betyaakov Sinagogu yakınlığında saat 09.30'da meydana gelen patlamalar.	istanbul 15 kasım 2003 tarih kuledibi neve şal şişli bet sinagog yakın saat 09 30 meydan gelen patlama	88.89%	“Şalom” and “Betyaakov “ are not Turkish words and don't take place in Turkish dictionary. Lemmatizer returns latest found lemma.
Sakıp Sabancı'nın 10 Nisan 2004 saat 05.55 sıralarında vefat etmesiyle ilgili dokümanlar.	sakıp sabancı 10 nisan 2004 saat 05 55 sıra vefat etme ilgili doküman	100%	
MGK toplantısında Cumhurbaşkanı Sezer'in Başbakan Ecevit'e anayasayı fırlatmasıyla gelişen olaylar.	mgk toplantı cumhurbaşkanı sezer başbakan ecevit anayasa fırlatma gelişen olay	100%	
Kıbrıs'ta açılan yeni üniversitelerin ve burada okuyan öğrencilerin sorunları, nasıl öğrenci aldıkları, denklik, kalacak yurt, öğretim üyesi bulma konusunda yaşanan sorunlar.	kıbrıs açıl yeni üniversite okuyan öğrenci sorun öğrenci al denklik kal yurt öğretim üye bulma konu yaşan sorun	100%	
Türkiye'ye 2003'te gelen turist sayısı ve dağılımı, illerdeki turizm durumu, turizmin ekonomiye katkısı,	türkiye 2003 gelen turist sayı dağılım il turizm durum turizm ekonomi katkı	100%	
Türkiye'nin Nükleer santral çalışmaları, nükleer santral projeleri	türkiye nükleer santral çalışma nükleer santral proje	100%	
hızlı tren kazasının nedenleri ve alınan önlemler	hızlı tren kaza alın önlem	100%	
Yüksek Öğretim Kurulu ,	yüksek öğretim kurulu yok	92.85%	The lemma of

YÖK'ün kuruluşu, üniversitelerimiz üzerindeki olumlu olumsuz etkileri, eleştirilen yönleri, YÖK hükümet ilişkileri	kuruluş üniversite olumlu olumsuz etki eleştiril yön yök hükümet ilişki		“kurulu” must be “kurul” (commision) But our lemmatizer returns “kurulu” (installed)
İbrahim Tatlises'in yaşadığı aşklar ve kadınlarla ilgili yarattığı huzursuzluklar kavgalar.	ibrahim tatlı yaşat aşk kadın ilgili yarat huzursuzluk kavga	77.78%	“Tatlises” is a special name doesn't have a specific lemma.The lemma of “yaşadığı” must be “yaşa” + “(mak)” (to live) But our lemmatizer returns “yaşat” + “(mak)” (to keep alive)
Parçalanmış aile bireylerinin yaşadığı sorunlar, özellikle bu türden ailelerin çocuklarının ve kadınlarının durumları.	parçalan aile birey yaşat sorun özellikle tür aile çocuk kadın durum	90.91%	.The lemma of “yaşadığı” must be “yaşa” + “(mak)” (to live) But our lemmatizer returns “yaşat” + “(mak)” (to keep alive)
Aile bireyleri arasında yaşanan şiddet olayları ve sebepleri. Çocuklara ve kadınlara uygulanan şiddet, buna maruz kalanların yaşadığı sorunlar.	aile birey ara yaşan şiddet olay sebep çocuk kadın uygulan şiddet maruz kalan yaşat sorun	93.33%	.The lemma of “yaşadığı” must be “yaşa” + “(mak)” (to live) But our lemmatizer returns “yaşat” + “(mak)” (to keep alive)
Türkiye’de son yıllarda özellikle Karadeniz bölgesinde arttığı düşünülen kanserli hasta sayısının Cernobil olayı ile varsa olan ilişkisi ve bu ilişkiyi irdeleyen çalışmalar, resmi kuruluşlar tarafından verilen istatistiklerin güvenilirliği.	türkiye son yıl özellikle karadeniz bölge art düşünül kanserli hasta sayı ce olay var ol ilişki ilişki irdeleyen çalışma resmi kuruluş tarafından veril istatistik güvenilirlik	96%	“Çernobil” is not a Turkish word and therefore doesnt take place in Turkish dictionary. Lemmatizer returns latest found lemma.
Futbolda yaşanan şiddet olayları, bunların nedenleri ve engellenmesi için alınacak önlemler.	futbol yaşan şiddet olay engellenme alın önlem	100%	
Türkiye’de son yıllarda ikinci el otomobil piyasasındaki durum, son dönemlerde piyasada	türkiye son yıl ikinci el otomobil piyasa durum dönem piyasa yaşan düşüş	100%	

yaşanan düşünün sebep ve sonuçları, yeni otomobil piyasasındaki yeniliklerle bağlantısı	sebep sonuç yeni otomobil piyasa yenilik bağlantı		
Türkiye'den kaçırılan tarihi eserler ve tarihi eser kaçakçılığı karşı yapılanlar	türkiye kaçır tarih eser tarih eser kaçakçılık karşı yapılan	100%	
İnsanların eğlenmesi ve kültür paylaşımı yapabilmesi için düzenlenen festivaller.	insan eğlenme kültür paylaşım yapabilme düzenlen festival	100%	
Türkiye'de bayram tatillerinde meydana gelen trafik kazalarının nedenleri, ve alınan önlemler.	türkiye bayram tatil meydan gelen trafik kaza alın önlem	100%	
öğrenmeyi etkileyen faktörler ve etkileri, öğrenme teknikleri	öğrenme etkile faktör etki öğrenme teknik	100%	
Kekik otunun faydaları, sağlık üzerindeki etkileri	kekik ot fayda sağlık etki	100%	
Türkiye'de telif hakkı yasalarının durumu ve bu konuda yapılan çalışmalar	türkiye telif hakkı yasa durum konu yapılan çalışma	100%	
İnternet'in yaygınlaşması, sunulan hizmetler, toplum üzerindeki etkileri.	internet yaygınlaşma sunul hizmet toplum etki	100%	
Türkiye'de tarım ve hayvancılık alanında yaşanan problemler ve bunların çözüm yolları.	türkiye tarım hayvancılık alan yaşan problem çözüm yol	100%	
İran'ın nükleer enerji ile ilgili politikaları, açıklamaları, nükleer enerji ile ilgili İran'da sürdürülen faaliyetler, uluslararası toplumdan İran'a nükleer enerji politikaları ile ilgili yöneltilen tepkiler veya verilen destekler	iran nükleer enerji ilgili politika açıklama nükleer enerji ilgili iran sürdür faaliyet uluslararası toplum iran nükleer enerji politika ilgili yöneltilen tepki veril destek	100%	
Satrancın yazılı basında ne ölçüde yer aldığı	satranç yazılı basın ölçü yer al	100%	
Genlerin insan sağlığı üzerindeki etkisi, hastalıkların kalıtsal nedenleri.	gen insan sağlık etki hastalık kalıtsal	100%	

hiperaktivite ve dikkat eksikliği nedir Belirtileri, teşhisi, tedavisi nelerdir Çocukların ve yetişkinlerin günlük yaşamına olumlu ve olumsuz etkileri nedir Hiperaktif çocuklara öğretmen nasıl yaklaşmalı Bu çocuklara yönelik eğitim sistemi nasıl geliştirilebilir	hiperaktivite dikkat eksiklik belirti teşhis tedavi çocuk yetişkin günlük yaşam olumlu olumsuz etki hiperaktif çocuk öğretmen yaklaşma çocuk yönelik eğitim sistem geliştir	100%	
Türkiye'deki lenf kanser istatistikleri	türkiye lenf kanser istatistik	100%	
28 Şubat süreci ve Türkiye üzerindeki etkileri	28 şubat süreç türkiye etki	100%	