Automatic Spelling Mistake Corrector Using python GPGPU Parallelism

K.D.K. Shamantha
Sri Lanka Institute of Information Technology, Malabe, Sri Lanka.

To autocorrect large text document, take several processing techniques and considerable amount of time. Because need to check each and every word in the document and if need to correct word, must correct. In word processing software shows misspelled word but user need to correct it manually by selecting correct word from given set of words. This project make user to correct spelling for given text document automatically. Using rule based approach each correction suggestions go through a processing to get the most suitable correction word with probability. Using Pycuda, Natural Language Processing and General Purpose Graphical Processing Unit programming reached the target with much efficiency in reading and correcting words, that cannot get from single processing mechanism.

Key words: Automatic, Rule-Base, Natural Language Processing, General Purpose Graphical Processing Unit

I. INTRODUCTION

Word spell checking is well researching area in natural language processing. It deals with spelling error detecting and automatic correcting of errors in an electronic word (text) document. Auto correcting are widely applicable for tasks like word-processing and post-processing. There are several approaches to spelling correction have been discovered in the literature, use rule-based, dictionary-based, statistical, etc.

To autocorrect large text document, take several processing techniques and considerable amount of time. Because need to check each and every word in the document and if need to correct word, must correct. In word processing software shows misspelled word but user need to correct it manually by selecting correct word from given set of words.

Spellchecking is a task of predict that which words in a text document are misspelled. These drops may be presented to a reader by highlighting the misspelled words in the text. Spelling correction is the task of replacing the correction hypotheses for words misspellings. In this paper discussed, implement and evaluation, achieves spellchecking and auto correction.

Ideally, parallel processing makes mistake search and correction much faster because subtasks of auto correction can processing as separate CUP executions. In practice, it is bit difficult to divide a program into subparallel tasks because of dependencies among them. Serial search may take considerable amount of time for a large text document.

CUDA is a parallel computing platform and programming model invented by NVIDIA. Most parallel programming techniques are moving to programming with CUDA [1]. CUDA is more It allows increases in computing performance by harnessing the computational power of the NVIDIA graphics processing unit (GPU). CUDA can handles hundreds of multiple threads in a single execution.

Therefore, CUDA makes most processing algorithms much faster than in serial processing. Following approach can improve execution power and reduce the result time.

II. STATE OF THE ART

Spell checking and Spell correction are among the oldest text processing problems, and there are many different solutions have been proposed since 1992. Most approaches are based on the use of more manual compiled resources. That many areas of natural language processing(NLP), the spelling autocorrecting have been increasingly experiential [2].

The shortest approach is to model the causes of spelling mistakes directly, and convert them into an algorithm or an error model.

N-gram study is defined as a method to find misspelled words in a huge text document. Instead of comparing each whole words in a text document to a given dictionary. A checking is done using an n-dimensional matrix array where actual n-gram rates were stored. If a non-existent or a rare n-gram and if it is found the word is highlighted as a misspelled word [3].

In Dictionary Lookup methodology, Dictionaries can be represented in many different ways, with their own features like speed and for storage requirements. Larger dictionaries may be a dictionary with most common words combined with a set of extra dictionaries for a specific subject. These dictionaries also take more space and time to search. The non-word mistakes can be detected as said above by going through each word against the dictionary. Too small a dictionaries can give the user many incorrect rejections of correct words and too large can accept as high number of valid whit low-frequency words.

Most present spelling correction methods target on isolated words, without taking into reason the written context in which the string appears.

Rule-based methods they work by having a set of predefined rules that capturing common spelling mistakes and applying rules to the error words. Naturally these rules are opposites of common errors. Each correct word generated by the process is

taken as a suggestion. The rules base also has probability. That by ranking the suggestions by gathering the probabilities for the applied rules [4].

Using neural networks can also make a possible solution technique. But it looks like it has to complete a bit more before that it can be used general purpose. Current methods are based on back-propagation by using one output node for each word in the dictionary. An input node for every possible n-gram in every location of the word. N can be usually being one or two. Usually only one of the outputs should be active showing which words the network proposes as a correction.

SpellNet explained spelling mistakes are roughly divided as nonword errors (NWE) and real word errors (RWE). If the misspelt string text is a valid word string in the English language, then it is called as an RWE, else it is an NWE. For example, in English, the word "fun" might be misspelt as "pun" or "bun", although the previous is an RWE, the word latter is a case of NWE. It is very easy to notice an NWE, but mistake correction procedure is non-trivial. RWE, bit difficult to detect as it appropriate syntactic and semantic analysis of the given text, though the very difficult of correction is similar to that of NWE [5].

Bangla Spell Checker can work as in both offline and online. It has a graphics user interface(GUI). Whenever the user types bangle text in real-time it checks for wrong spelling word and gives suitable suggestion. For a single error word. The correct word can found within top suggested 4 words in the suggestion list. Words having more than 1 error are so taken. But Suggestions cannot be given on some modified words. And It has ability to add new words into the dictionary against which spellings are check [6].

Automatically trainable system was the arithmetic model for spelling errors replaces verbal knowledge with a training amount of misspelling errors, which was compiled by hand. This approach has also been stretched to include a pronunciation model.

METHODOLOGY

In this paper describes implementation using rule base module. For Spelling correction process the basic functionalities can identified as follows.

i. Automate word auto correction for given text document for text domain information.
ii. Implement parallelize analysis and correction word function.
iii. Provide final output with accurate and efficient result.

User input text document should be in the English language. For serial execution processing designed for Peter Norvign [7] Proposed Model for spelling correction.

CUDA used as general purpose computations using GPU. CUDA can handle massive parallelism defining many hundreds of threads.

In this study used python library PyCUDA for implementation. Also including python numpy libraries.

In this paper CUDA 7.5 [8] and PyCUDA 2016.1.2[9] installed for computations.

Following figure explained the flow of the application process.

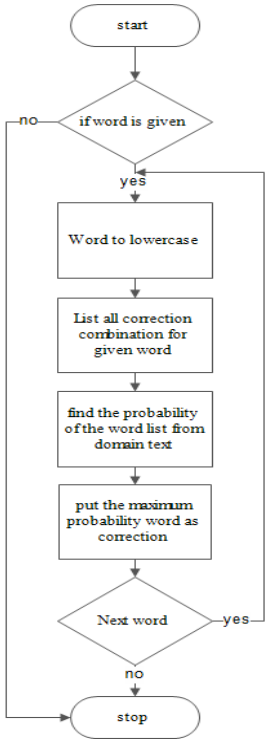

Figure 1. High-level diagram of the system

According to the probability given theory explained approach to correction with given word (w) and correction (c) out of all possible applicant corrections. Maximum probability of the 'c' is the correct word for word 'w'.

Argmax c E Candidates P (c | w)

Argmax c E Candidates P (w | c) (P(c) / P(w))

{Bayes' Theorem}

P(w) -> 1 {because for given word}

Argmax c E Candidates P (w | c) P(c)

P(c) = probability of correct word showing in the analysis text document.
P(w|c) = if probability of giving correct word 'c', probability of having word 'w'

For given word the might be different correction approaches.

i. letter deletion
ii. letter replace
iii. letter transpose
iv. new letter insertion

For the given correction word, find all the combinations of the word can represent.

In deletion get the list of words deleting each and every letter at given time. For example, given word is "suset". Output looks like ['uset','sset','sust','suse','suet']. For deletion here going to split the word into two parts making easy accessing to making the word. Two parts are letter starting from first to last dividing from a particular letter in the word. For example, word 'suset' makes [('', suset), ('s','uset'), ('su','set'), ('sus','et'), ('suse','t'), ('suset','')]. Then left-side and right-side combine in a suitable manner making deletes a letter from the word looks like in the word set output for deletion.

```
FOR i BETWEEN 0 AND word_length
    LEFT_SIDE = word [0] TO word [i]
    RIGHT_SIDE = word [i+1] TO word [word_length]
ENDFOR

FOR i BETWEEN 0 AND word_length
    LEFT_SIDE + RIGHT_SIDE [FROM i+1]
ENDFOR
```

Transpose makes swap between position of two letter in the word. For example, 'suset' gives ['usset','ssuet','suest','suste'], words count is word size -1. In here also can use the splitting methodology used in the deletion part.

```
FOR i BETWEEN 0 AND word_length
    LEFT_SIDE = word [0] TO word [i]
    RIGHT_SIDE = word [i+1] TO word (word_length)
ENDFOR

FOR i BETWEEN 0 AND word_length
    LEFT_SIDE + RIGHT_SIDE [0] + RIGHT_SIDE [TO word_length FROM 1]
ENDFOR
```

For replacement replace different letter for given letter in particular position in the word. Here all alphabet letter use to replace with given letter. Example for letter 'a' output as follows for 'suset', ['auset', 'saset', 'suaet', 'susat', 'susea']. For all the letters gives 5(size of the word) * 26 (alphabet letters) = 130.

Insertion new latter make the same as explained above. Inserting new letter between two letters. In here for all the letters in the alphabet are used between letters. For letter 'a' example 'suset' gives ['asuset', 'sauset', 'suaset', 'susaet', 'suseat', 'suseta']. In here there are 6(size of the word + 1) * 26(alphabet letters) = 136 word can have.

Insertion and replacement function suggested parallel processing using CUDA Kernel function. Number of combination getting high when the size of the word is high. And for each letter in the alphabet need to do the same task. That make single instruction multiple data (SIMD) methodology. Parallel execution runs in 26 threads (alphabet size). Here need have 2D array of char, size of word_lenght ^ 2. Following Pseudo code show a single thread execution in replacement function.

```
idx = thread_id
letter = letters(idx)
cells_size = word_length ^ 2
start_point = idx * cells_size
end_point = (idx+1) * cells_size

FOR i = start_point TO 0 AND end_point AND i = i+(word_length+1)
    word(i) = letter
ENDFOR
```

Following Pseudo code show a single thread execution in insertion function. Here defined array to store insertion data.

```
idx = thread_id
letter = letters(idx)
array [word_lenght^2]
counter = 0

FOR i BETWEEN 0 AND word_length+1
    FOR j BETWEEN 0 AND word_length+1
        IF i == j
          array [(word_lenght*i) +j] = letter
        ELSE
          array [(word_lenght*i) +j] = word[j]
          counter++
        ENDIF
    ENDFOR
    counter = 0
ENDFOR
```

After getting all the combinations (deletion, replacement, insertion, transpose), need to find out the probability with the domain text. The accuracy makes high if number of words in the text domain is high. So in here need to search through each word in the domain text document. According to the above example gives 275 word list. Each word need to search through the domain text document. For this mechanism suggest CUDA search function.

For searching function, we define single thread for single character in the text domain. In the kernel function we mention staring from thread index search the word in the domain text and return the staring position of the domain text. By counting number of returning position can find the probability of the word.

Following Pseudo code show a thread execution of the kernel.

```
idx = thread_id
domain_text[text_size]

IF domain_text[idx] = = word [0]
    FOR i BETWEEN 0 AND word_length
        IF domain_text[idx+i] ! = word[i]
            BREAK
        ELSE
            RETURN
        ENDIF
    ENDFOR
ENDIF
```

III. RESULTS AND DISCUSSIONS

The values for time in sequential execution and parallel execution clear that CUDA does what it possibilities and it gives a benefit in terms of execution times when compared to the sequential execution. When considering replacement and insertion algorithms, does not give efficient result with CUDA that showing CUDA is not preferable for simple parallel execution. Where search function speed-up the execution achieving high effective manner.

However, this paper was proposed for theoretical work, and used Python programming language, which is a just in time(JIT) interpreted language.

The parallel part, does not seem to show any sign of revolution in performance, when executing with different size of element. This may be cause of architecture of CUDA in NVidia CUDA, which take in highly memory. To get maximum productivity from CUDA parallelism, the array sizes must be a very large value. CUDA timings of the parallel execution would be according to the complexity that of serial execution time.

Theoretically the norvig proposed probability theory can achieve spelling corrector 80% to 90% accuracy. But in the implementation could achieve 68% accuracy. But it speeds up the execution that word processing speed in huge.

Advantages

i. Detects most of the spelling errors and typing errors.
ii. Better accuracy.
iii. Save more time
iv. Can handle huge data set.

Disadvantages

i. Does not separate homonyms
ii. Recognizes words as mistakes if they are any word spelled using a language form different than the default word in the program.

In this project have to implement the automatic spelling correction application. The development this application by using python PyCUDA technology. Python's numpy library with very powerful at array operations. Intel 6nd generation i7 4-core 8 thread CPU and Nvidia GTX950m GPU is used for this experiment. Different test with more powerful hardware can accomplish larger array data sets, to reproduce the computational complexity for time. GPU with additional memory space expects decrease in the execution time of the code, because of larger array sizes. And that described by the corresponding theoretical complexity given earlier.

IV. CONCLUSION

This paper the results verify there is amount volume of speed-up can be accomplished using CUDA parallelism instead of a serial code execution. To get the maximum output of the GPU processing power, need to restructure the algorithms according to the underlying parallel GPU architecture of the graphics cards.

ACKNOWLEDGMENT

This study is supported by Sri Lanka Institute of information Technology and parallel computing module lectures and labs conducted by Mr Darshika expressing heart left thankfulness and to all those who have generously offered their ideas towards the completion of the module project.

REFERENCES

[1] "Parallel programming and computing platform," NVIDIA, [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Accessed 20 Sep 2016].

[2] N. Gupta and S. N. Gupta, "Spell Checking Techniques in NLP: A Survey," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 2, no. 12, pp. 217–221, Dec. 2012.

[3] Dr. R.K Sharma ,"The Bilingual Punjabi English spell checker ," Resource centre for Indain language Technology Solution ,TDIL newsletter.

[4] C. Whitelaw, B. Hutchinson, G. Chung, and G. Ellis, "Using the Web for Language Independent Spellchecking andAutocorrection," *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pp. 890–899, Aug. 2009.

[5] M. Choudhury, M. Thomas, A. Mukherjee, A. Basu, and N. Ganguly, "A Cross-linguistic Analysis through Complex Network Approach," 2007.

[6] "Baraha - Indian Language Software," in *baraha*, 1998. [Online]. Available: http://www..Baraha.com. Accessed: Sep. 22, 2016.

[7] P. Norvig, "How to write a spelling Corrector," in *norvig*, 2007. [Online]. Available: http://norvig.com/spell-correct.html. Accessed: Sep. 19, 2016..

[8] "CUDA Toolkit," in *developer nvidia*, NVIDIA Developer, 2016. [Online]. Available: https://developer.nvidia.com/cuda-toolkit. Accessed: Aug. 16, 2016

[9] A. Kloeckner, "PyCUDA 2016.1.2 documentation," in PyCUDA, 2008. [Online]. Available: https://documen.tician.de/pycuda/. Accessed: Sep. 20, 2016.