Accelerated Spam Filtering with Enhanced KMP Algorithm on GPU

Venkata Krishna Pavan Kalubandi
School of Information Technology
Vellore Institute of Technology
Vellore, India.
krishnakalubandi@gmail.com

Varalakshmi M.
School of Information Technology
Vellore Institute of Technology
Vellore, India
mvaralakshmi@vit.ac.in

Abstract— **Spam filtering is one of the most important applications in email services that has become increasingly sophisticated due to the enormous usage of Internet. Traditionally, spam filters have been implemented on the CPU with a pattern matching algorithm. In this paper, an accelerated spam filtering mechanism that uses GPUs is presented. The filtering process utilizes an enhanced version of Knuth Morris Pratt pattern matching algorithm that outperforms the serial versions up to 12x and also performs more efficiently compared to other parallel versions. The parallel algorithm is to develop and advanced keyword based Naïve Bayesian classifier speeds up the spam filtering up to 2 times compared to CPU.**

Index Terms— **Spam Filtering, Pattern Matching, KMP, Bayesian Spam Filters**

I. Introduction

With the advent of the Internet, email has emerged as a major form of communication not only for users but also for companies as a form of marketing their products. The bulk email messages sent by the companies that advertise products that are unnecessary or irrelevant to the users are termed as “spam” emails. Various types of filter based algorithms have been introduced to categorize email into spam or ham. [1] Spam filters include keyword based detection, statistical based algorithms [2], image based algorithms [3] and so on. Many spam filters employ some sort of pattern matching algorithm in order to detect keywords or find patterns in the email. Pattern Matching is an important study in Computer Science and has its roots in the fields of Molecular Biology [4], Intrusion Detection [5] and Bio Sequencing [6]. Various algorithms have been proposed in this field and the number continues to increase. With the increasing computing capabilities and increasing volumes of data, efficiency of these algorithms has become critical and this calls for numerous modifications to existing algorithms and proposal of new algorithms that suit a particular domain.

With the advent of GPU’s, parallel programming approaches have received a good attention of the research community. GPU Computing involves harnessing the power of GPU’s along with the CPU to tackle scientific problems for engineering and enterprise applications. Introduced by NVIDIA in 2007, GPU’s are in use in wide range of devices including mobiles, desktops, servers to accelerate the performance and improve efficiency. GPU’s have been put to good use in a variety of applications like Image Processing [7], Primality Testing [8], Linear Algebra [9] etc. GPU’s are also used in machine learning particularly in Neural Networks [10]. Due to cost effectiveness and high throughput GPU’s are receiving special attention and are being used in many applications.

In this paper, an enhanced version of Knuth Morris Pratt pattern matching algorithm [11] that divides the text to be searched into chunks of smaller texts and utilizes the power of GPU to search in parallel is proposed. The proposed algorithm also takes care of the boundary conditions implicitly with minimal overhead that makes it very fast to execute and can achieve speeds of up to 12x compared to CPU. The adaptability of the algorithm with large text sizes is also tackled by implementing overlapping data transfers in the GPU [12], that not only offers results asynchronously but also adds more parallelism to the execution thereby increasing the speed of searching. In order to maintain the best speeds over various GPU architectures, the algorithm varies the size of the smaller chunks according to the pattern size by following some heuristic based measures that are discussed in the subsequent sections.

The algorithm is then used to create a very powerful spam filter that classifies thousands of emails in seconds. It classifies an email as spam by looking at certain keywords and also updates the keywords by using Naïve Bayesian classification method. The rest of this paper is organized as follows. Section II provides a detailed literature survey of existing algorithms and their potential drawbacks. Section III provides an overview of background details including pattern matching, KMP algorithm, Spam Filtering, Naïve Bayesian classification and GPU Programming with CUDA. Section IV gives the details about the experimental methodology used to implement modified KMP algorithm. Section V presents the experimental results to demonstrate the speed-up of GPU based algorithm. Finally, Section VI draws conclusions and provides some insights for further research and innovation in the fields of pattern matching, parallel computing and machine learning.

II. Previous Works

Traditionally Naïve Bayesian spam filters are considered as the simplest yet effective spam filters [13]. Within Naïve Bayes, there are many types of decision making algorithms that use various probabilistic approximations [14]. Keyword based spam filters may not be effective in situations that require specific spam filter choices for every user. An excellent comparison of both the above approaches is given in [1]. The Naïve Bayesian Spam Filter certainly has its advantages; however, it is still susceptible to an attack called Bayesian Poisoning in which an attacker essentially injects an email with lots of legitimate words and his intended message in a highlighted manner [15]. The spam filter may incorrectly classify the email as ham creating problems to the user. In such cases a keyword based spam filtering method may be more effective as it looks only for a set of keywords. The spam filter implemented in [16], uses Naïve Bayesian Classification combined with Aho-Corasick algorithm for pattern matching. However, the classification reached only an accuracy of 70% which makes it suitable only for a limited range of applications.

Research shows that the idea of implementing pattern matching algorithms on GPUs is not entirely new. A modified version of Aho-Corasick algorithm [17] and various other algorithms (Naïve, Boyle-Moore-Horsepool, Quick-Search) including the Knuth Morris Pratt algorithm have been implemented [18] [19] [20]. Also, using the pattern matching algorithms in spam filtering is not entirely new. In [16], a parallel spam filtering algorithm is used based on Aho-Corasick algorithm. These existing implementations and their potential drawbacks are discussed in brief.

The parallel Knuth Morris Pratt algorithm implemented in [18] achieved a speed ranging between 5x to 12x. However, the speed up decreased for increasing pattern sizes and no mention has been made regarding dealing with the boundary cases. This may be partly attributed to not adapting to changing pattern sizes. The parallel algorithm implemented in [21] uses the idea of giving extra work load to each thread to check for the boundary conditions. Similarly, the algorithm implemented in [19] uses the approach of searching for the pattern twice to include boundary conditions. This increases the overall running time of the algorithm and also extra load on every thread.

In this paper, a new approach that uses both Naïve Bayesian and Keyword based spam filtering is used to accelerate the spam filtering process and also to overcome the attacks that are prevalent in Naïve Bayesian spam filter. The spam filter also has very good accuracy rate compared to the previous implementations.

III. Background

A. Pattern Matching and KMP Algorithm:

Pattern Matching involves searching for an occurrence of a pattern in a text of particular size. If the text is assumed as an array *T[1…n]* and the pattern as an array *P[1…m]* provided that $m \leq n$, a pattern P is said to be found in the text for a *shift s* if and only if *T[s + 1 …. s + m] = P [1…m]*. The pattern matching problem is a problem that involves finding all those shifts in a given text *T*. The Naïve String matching algorithm is discussed in brief in order to provide some intuition about KMP algorithm.

Naïve String-Matching Algorithm: The naïve algorithm finds all occurrences using a simple loop that checks the condition *P[1..m] = T[s + 1.. s + m]* for each *n - m – 1* possible values of *s*. This is basically a brute force search for all possible occurrences of pattern in a text. The algorithm runs in *O(nm)* time to find all occurrences. Since this algorithm times out for large n and m, it is not useful for practical purposes.

Knuth Morris Pratt algorithm: KMP algorithm is a plausible approach to the above problem, as it searches for the patterns in linear time. KMP algorithm utilizes a prefix function π that contains information about how the pattern matches against shifts of itself. This information can be utilized to avoid testing useless shifts that occur in the naïve pattern matching algorithm.

The pre-computed function is defined as follows. Given a pattern *P[1..m],* the *prefix function* for the pattern *P* is the function $\pi : \{1,2,\dots,m\} \rightarrow \{0,1,\dots,m\text{-}1\}$ such that

$$\pi\, [q] = max\{k{:}k<q \text{ and } Pk \text{ is a proper suffix of } Pq\}$$

The calculation of prefix function is illustrated in Table.1. The complexity of the algorithm is *O(m + n).*

Table 1 Prefix Function of KMP Algorithm

I	1	2	3	4	5	6	7	8
P[i]	*a*	*B*	*a*	*b*	*a*	*c*	*B*	*a*
Π[*i*]	0	0	1	2	3	0	0	1

Therefore, by avoiding unnecessary re-checking of parts of text, the KMP algorithm easily outperforms the Naïve string matching algorithm and is considered as the fastest pattern matching algorithm for a given alphabet size. [11] [22]. However, for a huge text, it still takes considerable amount of time to find all the patterns. This can create problems especially in large email servers that need to compare huge amount of text and look for keywords to identify whether an email is considered spam. There is a scope for reducing the run time of the algorithm further by harnessing the power of GPU's which is the aim of this paper and will be elaborated in the further sections.

B. Spam Filtering and Naïve Bayesian Classification:

Spam filters can be generally classified into two major types.

Keyword Based Spam Filters: In keyword based spam filtering each incoming email message is compared with a set of keywords that are generally associated with spam. **E.g.:** Viagra, sex, free etc.

Statistical Based Spam Filters: Statistical based spam filters employ some sort of pre-computation on the existing emails and then use the data obtained to classify a new email as spam or ham. *E.g.:* Naïve Bayesian Spam Filtering.
Among statistical based filters, Naïve Bayesian classifier is the most common and commercially available spam filter.

Naïve Bayesian Spam Filter: Fine-tuned by Paul Graham [23], Naïve Bayesian spam filter is a popular spam filtering technique. It uses the words contained in the email as a decision parameter to detect whether an email is spam or not. Generally, a decision is based on the bag of words approach where number of occurrences of a word in an email and the probability of the occurrence of such words decide whether an email is spam or not.

Naïve Bayesian Classification begins by treating an email E as a set of words $w_1, w_2, \ldots, w_n$. Then, the problem of spam filtering can be defined as the "the posterior probability $P(E|S)$ that an email containing words $w_1, \ldots w_k$ is spam given prior probabilities $P(w_1, .., w_n|S)$ of emails that are spam that contained words $w_1, \ldots w_k$ and an evidence of an email to be spam". It is impossible to calculate the relative probabilities of combinations of words; hence, a naïve assumption is introduced into the equation that provides conditional independence. The probability of occurrence of a word w_i is independent of the probability of occurrence of any other words in the set W [13].

$$\begin{aligned} P(E|S) &= P(w_1, \ldots, w_n|S) \\ &= \prod_{i=1}^{n} P(w_i|S) \end{aligned}$$

The Naïve Bayesian Classifier is trained with these probabilities from a known dataset. Once the probabilities are built, the classifier can look at email it hasn't seen before and calculates the probability of it being spam by multiplying the probabilities of the words being spam contained in the email and the probabilities of the words not being spam. A higher probability (threshold) level is set and then the decision is made to classify an email as spam or ham.

C. GPU Programming with CUDA:

CUDA is introduced by NVIDIA to facilitate parallel programming. CUDA is offered as an application program interface that allows developers to use programming languages like C, C++ to build massively parallel programs easily. Some of the key terms involved in CUDA are described in brief below [24].

CUDA Kernel: CUDA C is a wrapper around C that allows to define functions in C, called *kernels*, which, when called, are executed N times in parallel by N different *CUDA threads* inside the GPU, as opposed to only once like normal C functions. Each thread that executes the kernel is given a unique thread ID that is accessible within the kernel through the built-in *threadIdx* variable [24].

Threads and Memory: The CUDA threads can access the data from multiple memory spaces during their execution process. Every thread has its own private local memory. Every thread has access to the shared memory space and this memory is called shared memory. The life time of a shared memory is limited to the block scope. Every thread also has access to the global memory that lives throughout the execution [25].
Block: A block consists of a group of threads. The order of the execution of threads within a block is not fixed – they could execute concurrently or serially and in no particular order. When some of ordering required with in the block, explicit synchronization constructs can be provided that suspends the progress of a thread until all the other threads finish their progress.

Grids and Streams: Grid is a group of blocks. There's no synchronization between the blocks. A *stream* in CUDA is a set of instructions that run on the device in the same order as given in the host code. Operations inside a stream get executed in the same order; however, operations in different streams can be interleaved and, they can be run concurrently [12]. In this paper, this feature of streams is used to further speed up the pattern matching process.

IV. METHODOLOGY

The methodology is divided into two sections. They are described in detail below.

A. Methodology for evaluating Parallel KMP Algorithm:

In order to achieve the best performance, the following approach has been identified and implemented. For small patterns, each thread takes a chunk of the text with size up to 30 times the pattern. For the same text size but with a larger pattern size, each thread takes a chunk of the text with up to 15 times the pattern. Thus the text is divided into different parts based on the pattern size as opposed to [21], where fixed size was used. This plays a key role in the execution time of the algorithm because the work done per thread varies according to pattern size and it has an impact on the total number of threads. The correct multiplier for each pattern size is dependent on the architecture of the underlying GPU. Each thread is then given a chunk of the text. The chunk size and total number of threads can be calculated as follows

*chunk_size = multiplier * pattern_size* (1)

*tot_threads = text_size / (multiplier * pattern_size)* (2)

where multiplier is determined based on pattern size and GPU architecture. To avoid missing a pattern split over two parts of text, the following approach is used. Each thread along with KMP algorithm, computes the number of matches that align with the last 1 to m-1 characters of the pattern and number of matches that align with first 1 to m-1 characters of the pattern. The number of matches found to the left and right of the chunk is stored in a pre-allocated look up table. At the end of execution, the stored values are compared and number of matches at the boundaries is calculated. This entire execution takes place in a single loop to improve performance. The pattern and the prefix function are directly copied to the GPU at a single stretch. The prefix function is computed on the CPU to increase the overall performance of the algorithm.

Algorithm:

ParallelKMP(start, end, text, pattern, table, result)

```
Start – Starting position of the data in text array
End – Ending position of the data in text array
Text – Haystack
Pattern – Needle
Table – Stores boundary matches
Result – An array to store the positions of the matches

Begin
j = start, k = end-1; m = pattern.size(), n = 0;
left = 0, right = 0, count = 0;
for( i = start; i < end; i++) {
        //do regular kmp search
        //if match
        count++;
        //boundary case
        x =  m – (n + 1);
        if pattern[n] == text[j] { left++, j++ ;}
        else if(left) { left = 0; j = start; }
        if pattern[x] == text[k] {right++, k--;}
        else if(right) { right = 0; k = end-1;}
        n++;
}
table[threadIndx* 2] = left
table[threadIndx* 2 + 1]  = right
result[threadIndx] = count
End
```

The time complexity of the parallel and serial algorithms can be summarized as follows. Computational complexity of the *Serial Version* turns out to be $O(m + n)$ for the text size, n and pattern size, m. Its parallel counterpart takes $O(n / T_n + c)$ time with T_n to be the total number of threads as given by (1) and c, the overhead constant that depends on the architecture. Hence, this algorithm has an overall better time complexity when compared to

$O(n / Tn + (2n\text{-}2) + c)$ achieved in [21]
and
$O(n / Tn + m - 1)$ achieved in [19]

Since each thread stores two integer values to calculate boundary values, the look up table requires a memory of $O(Tn * 8)$ *bytes*. This memory space could be further decreased by using a dynamic vector based array, using the boost library, so that only potential matches are stored and compared.

B. Methodology to evaluate Spam Filter:

The modified parallel version of KMP algorithm mentioned above is further streamlined to facilitate faster pattern matching. The algorithm is deliberately allowed to leave out the matches occurring in the boundary cases as missing a highly unlikely occurrence has least effect on the classification of spam or ham. The algorithm is modified to include searching for multiple patterns with a single kernel call. These modifications include combining prefix functions, patterns and minimizing memory copying from GPU to CPU and vice versa. They are described in detail below.

Multiple pattern checking with a single kernel call: This process involves preprocessing the patterns and prefix functions for each pattern. For a set of patterns $P_1, P_2...P_n$ the following process is employed

1. An array P is constructed from $P_1, P_2...P_n$ such that
   $P = P_1 \,||\, P_2 \,||....||\, P_n$ where || denotes concatenation operator
2. Now, for every pattern (n patterns) P_i construct a prefix function π_i as described in section II and construct an array π such that
   $\pi \;||\!= len(P_{i-1}) + \pi_{i,j}$ *for j in* π_i *for i in n*
   with $len(P_{-1}) = 0$, *|| being concatenation operator*
3. Next, the individual offset for each pattern must be constructed in order to facilitate multiple pattern matching. Construct an array O of length n such that
   $O[i] = len(Pi)$
4. Finally, construct an empty array *res* that stores the results of number of occurrences of each pattern that is same as the array in single pattern matching algorithm. The size of the array *res* will be dependent upon number of patterns and number of threads used to execute the algorithm.
5. The similar offsets are constructed back in the kernel to get back the prefix function of each pattern, the pattern and the storage of count of each pattern for every thread.

The text, the offsets in the text, number of threads, chunk of text for each thread is same as the single pattern version. The modified kernel code can now be used in bag of words based Naïve Bayesian classification, as it gives number of occurrences for each pattern in a text. Two types of classification are used. They are described in brief.

1. Bag of words for every word:

In this approach, it is assumed that nothing about the data other than spam or ham is known. The classifier is built from taking occurrence of each and every word and calculating the relative probabilities using Bayesian theorem. The probability for spam or ham is then compared and final decision regarding the email is made. In this scenario, there are two important points to be considered

1. **Data Cleaning:** Before counting the words, data cleaning and pre-processing have to be done on the text. There will

be words like “price”, “prices” that are same in meaning. String tokenizers combined with lemmatization should be used in order to identify and process such words.

2. **Stop Words:** Some common English words like “a”, “an”, “here”, “there” have large impact on the decision if they are considered. As these words cannot be used to predict an email is spam or not, pre-processing on each and every email must be done in order to remove these stop words.

The above process takes considerable time as the size of the emails and number of email increases as the algorithms are based on Natural Language Processing [26] [27]. However, if every word is considered, a powerful classifier with a greater accuracy can be built. Also, this type of classifier is vulnerable to Bayesian Poisoning attack [15].

2. Bag of words for a set of keywords:

In this approach, only most probable words that differentiate spam and ham are considered. For instance, the list given in [28], provides a list of words that occur in spam emails based on various categories like business, home based, commercial, medicine based and so on. The classifier can be built with these words and occurrences and whenever a new email arrives, the keywords and their probabilities can be matched to make a decision. This type of classification is very fast and is suitable for parallelization as it involves only pattern matching algorithm. It also withstands the poisoning attacks as it looks only for probable keywords and not every word. However, it may not be as accurate as the bag of words approach. Therefore, based on the application and level of detail vs time one of the two approaches can be used.

In this paper, both the first approach is implemented on the CPU. The second approach is then further optimized to use the GPU to provide very fast spam detection that has an acceptable accuracy rate.

Two experiments performed are.

1. **Bag of Words – Every Word – Serial – CPU Version**
2. **Bag of Words – Keywords Only – Parallel – CPU + GPU Version**

The metrics considered for evaluation are the *run time of the classification* and the *accuracy of the classifier*. The experiments are repeated under various conditions to reduce any bias. The second experiment is the proposed method that accelerates the spam filtering process.

V. Results and Discussion

The experiments were performed on a workstation with Intel Core i7-5500U CPU running at 2.4GHz, and supported by 8GB of RAM and features an *Nvidia GeForce 840M* GPU (with 384 cores and 2GB DDR3 global memory). The KMP algorithm articulated in this paper is implemented using C++ programming language with CUDA API.

The experimental results and discussions are also divided into two sections to better understand the performance and efficiency of both the algorithm and its application

A. Results of Parallel KMP Algorithm:

The test cases required for the computation are generated as follows. A highly dense random string of 194MB is generated with alphabet size $\sum = 26$. Random substrings of size 94.36MB, 5.38MB are taken from the above string. Various pre-chosen patterns are distributed randomly across these three strings. Pattern lengths used for searching are of length m = 7, m = 34, m = 134, m = 495.

Overlapping Kernel Execution and Data Transfers:

Inorder to overlap data transfers and kernel execution the following conditions have to be met. The device must be capable of “concurrent copy and execution”. Nearly all devices with compute capability 1.1 and higher have this capability. The kernel execution and the data transfer to be overlapped must both occur in*different*, *non-default* streams. The host memory involved in the data transfer must be pinned memory [12]. In order to achieve the best performance, the given text is split across 2-4 streams and the transfer from host to GPU is done asynchronously. This overlapping data transfer has significant performance improvement over direct transfer of text followed in [18] [19] [21].

Fig. 1, shows the execution time of the algorithm across three text sizes and a constant pattern size. A speed of 12x times the CPU version is achieved. It can be inferred from the above data that the GPU based algorithm’s performance increases as the text size increases.

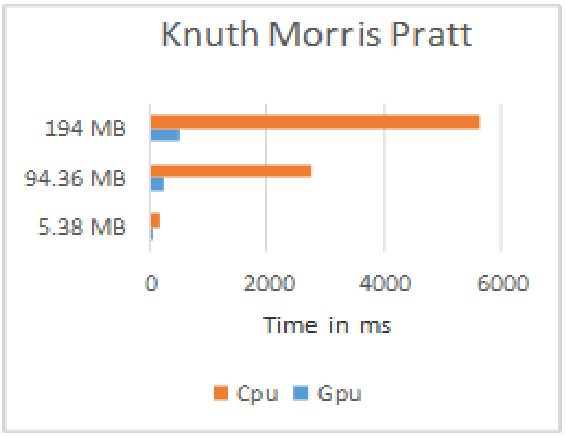

Figure 1. Knuth Morris Pratt CPU vs GPU

Fig. 2, shows the execution time of the algorithm across various pattern sizes and a constant text size. As the multiplier value changes for each pattern, a speed of 12x remained consistent with respect to CPU as opposed to the decrease in speed for large patterns in [18] [19] [21]. Thus, it is evident that a constant chunk of text size must not be preferred.

Fig. 3, shows the improvement in the GPU algorithm when overlapping data transfer is used with two streams. Even though the difference is minute, as the capability of streams in GPU increase, this optimization becomes crucial. Also when dealing with large text sizes and when intermediate results are important, the asynchronous transfer must be preferred.

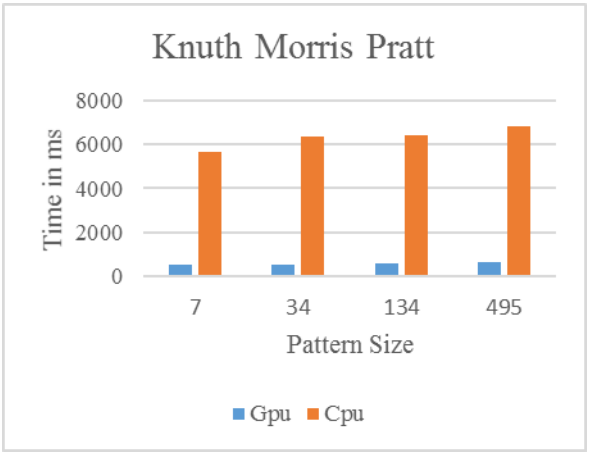

Figure 2. Knuth Morris Pratt for various pattern sizes

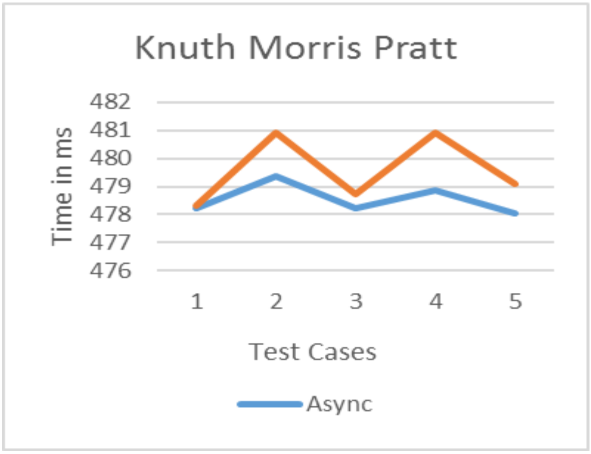

Figure 3. Knuth Morris Pratt GPU Synchronous Vs Asynchronous

B. Results of the Spam Filter:

NLTK and Pycuda:

The spam filter requires the usage of advanced natural processing algorithms that are beyond the scope of this research. Hence, the nltk available for Python is used to perform these tasks. NLTK, is toolkit available in Python that provides easy to use interfaces and consists of a basket of text classification libraries including classification, tokenization, lemmatising, parsing, and semantic reasoning [29] [30]. In this paper, the Naïve Bayes Classifier, tokenization, and lemmatisation algorithms are borrowed from the NLTK library.

PyCuda by A Klöckner, is an open source wrapper around CUDA in python that allows developers to access CUDA parallel computation API via python. The wrapper has many exciting features like object cleanup, automatic error checking and also offers speed that closely matches native CUDA [31][32] [33]. In this experiment PyCuda library is used the kernel code inside python so as to provide input to the classifier.

Enron Dataset: The enron dataset is a dataset consisting of 151 employees email records publicly made available by Federal Energy Regulatory Commission and put up online by Willam Cohen from CMU. A subset of that dataset enron1 is used as the core data set for this research. The dataset consists of total 5172 email of which 3672 ham emails of owner *farmer-d* and 1500 spam emails of owner *GP* [34].

Table 2 gives the detailed descriptions of both the experiments (serial – all words, parallel-keywords). Table 3 gives most informative features that had higher probability to be classified as spam or ham. As it can be seen from Table 2, the parallel version is **100%** faster than serial version with only a **7%** decrease in accuracy. By utilizing both the CPU and GPU, by implementing a fast parallel matching algorithm, it can be seen that execution time is greatly reduced. By including more probable keywords and by updating them periodically based on the user's choices, the parallel algorithm's accuracy can be further improved to match every word type. Hence, the CPU + GPU version can be effectively utilized in consumer oriented business server where speed of classification matters a lot and considerable accuracy is required. Only in case of very strict accuracy requirements, one needs to use to every word approach.

Table 2 Results of Spam Filter

Type	**Processor**	**Word Type**	**Run Time (s)**	**Accuracy**
Serial	CPU	Every Word	27.72	**92%**
Parallel	CPU + GPU	Keywords	**13.56**	85%

Table 3 Most Informative Features

Word	**Spam : Ham (ratio)**
Prescription	112.8 : 1
Spam	78.1 : 1
Cheap	70.7 : 1
Sex	67.5 : 1
Free	62.2 : 1
Viagra	58.5 : 1
Creative	49.7 : 1

VI. Conclusion and Future Work

In this paper, various existing parallel implementations of KMP algorithm were examined and a modified implementation of algorithm that achieves best performance in CUDA is presented. Both serial and parallel implementations were compared in terms of running time for different text sizes, pattern sizes and number of threads. It is shown that the parallel implementation of the algorithm is up to 12x faster than the serial implementation and it remained consistent with increasing pattern sizes. Overlapping data transfer is implemented to further improve the performance of the algorithm and to support large text sizes. Then, the algorithm is further streamlined to match multiple patterns in a faster way. It is used in a key word based spam filter using Naïve Bayesian Classification and the run times are compared with the serial version. The parallel spam filter completely outperformed the serial version with speeds up to 2x offering a good accuracy rate. This spam filter can be employed in servers that need to classify millions of emails with a blazing fast speed.

Future research on parallel processing and string matching could focus on further optimization of various string matching algorithms based on particular architecture. Prediction of the correct multiplier for a particular pattern size based on GPU architecture should be accounted. For text sizes that are very large and cannot be fit into the GPU memory at once, overlapping data transfer can be further optimized. Parallel pattern matching on GPU clusters can also be explored. Spam filtering can also be further extended to match sequence of patterns instead of only words to reduce Bayesian noise and also much more sophisticated classification models like Support Vector Machines and Neural Nets can be implemented. The accuracy of the existing algorithm can also be improved by striking a balance between the execution time and the required accuracy.

References

[1] I. Androutsopoulos, J. Koutsias, and K. Chandrinos, "An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages," *Proc. 23rd*, 2000.

[2] L. Zhang, J. Zhu, and T. Yao, "An evaluation of statistical spam filtering techniques," *ACM Trans. Asian Lang. Inf.*, 2004.

[3] G. Fumera, I. Pillai, and F. Roli, "Spam filtering based on the analysis of text information embedded into images," *J. Mach. Learn. Res.*, 2006.

[4] E. Rouchka, "Pattern Matching Techniques and Their Applications to Computational Molecular Biology-A Review," 1999.

[5] G. Vasiliadis, S. Antonatos, and M. Polychronakis, "Gnort: High performance network intrusion detection using graphics processors," *Int. Work.*, 2008.

[6] G. Mehldau and G. Myers, "A system for pattern matching applications on biosequences," *Comput. Appl. Biosci. CABIOS*, 1993.

[7] H. Patel, "GPU accelerated real time polarimetric image processing through the use of CUDA," *Proc. IEEE 2010 Natl. Aerosp.*, 2010.

[8] S. Khemnar, R. Chaudhary, and D. Kulkarni, "CUDA based Implementation of Parallelized Pollards Rho Algorithm for ECDLP: A Review," *Data Min. Knowl. Eng.*, 2015.

[9] J. Kurzak, P. Luszczek, M. Faverge, and J. Dongarra, "LU factorization with partial pivoting for a multicore system with accelerators," *IEEE Trans. Parallel*, 2013.

[10] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Adv. neural*, 2012.

[11] V. R. Knuth, D.E., Morris, J.H., & Pratt, J. H. Morris, Jr., and V. R. Pratt, "Fast Pattern Matching in Strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, 1977.

[12] M. Harris, "How to overlap data transfers in cuda c/c++," *Internet: devblogs. nvidia. com/parallelforall/how-*, 2012.

[13] T. Sun, "Spam Filtering based on Naive Bayes Classification," *Arch. Res. Pap. Babes Bolyai Univ.*, 2009.

[14] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam filtering with naive bayes-which naive bayes?," *CEAS*, 2006.

[15] J. Graham-Cumming, "Does Bayesian poisoning exist," *Spam Bull.*, 2006.

[16] S. Haseeb, M. Motwani, and A. Saxena, "Serial and Parallel Bayesian Spam Filtering using Aho-Corasick and PFAC," *Int. J. Comput. Appl.*, 2013.

[17] C. Lin, C. Liu, L. Chien, and S. Chang, "Accelerating pattern matching using a novel parallel algorithm on gpus," *IEEE Trans. Comput.*, 2013.

[18] X. Bellekens, I. Andonovic, R. Atkinson, and C. Renfrew, "Investigation of GPU-based pattern matching," *14th Annu. Post*, 2013.

[19] C. Kouzinopoulos and K. Margaritis, "String matching on a multicore GPU using CUDA," *Informatics, 2009. PCI'09. 13th Panhellenic*, 2009.

[20] M. Najam, U. Younis, and R. Rasool, "Multi-byte Pattern Matching Using Stride-K DFA for High Speed Deep Packet Inspection," *Sci. Eng. (CSE), 2014 IEEE 17th …*, 2014.

[21] A. Rasool and N. Khare, "Parallelization of KMP String Matching Algorithm on Different SIMD Architectures: Multi-Core and GPGPU's," *Int. J. Comput. Appl.*, 2012.

[22] T. Cormen, "Introduction to algorithms," 2009.

[23] P. Graham, "A plan for spam," 2002.

[24] N. Wilt, "The cuda handbook: A comprehensive guide to gpu programming," 2013.

[25] J. Shun-Liang, "GPU APPLICATION IN CUDA MEMORY," *Adv. Comput.*, 2015.

[26] J. Perkins, "Python text processing with NLTK 2.0 cookbook," 2010.

[27] M. Asghar, A. Khan, S. Ahmad, and F. Kundi, "Preprocessing in natural language processing," *Emerg. Issues Nat.*, 2013.

[28] K. Rubin, "The ultimate list of email spam trigger words," *Internet Mark. Blog| HubSpot*, 2012.

[29] S. Bird, "NLTK: the natural language toolkit," *Proc. COLING/ACL Interact.*, 2006.

[30] E. Loper, "NLTK: Building a pedagogical toolkit in Python," *PyCon DC 2004*, 2004.

[31] A. Klöckner, "PyCUDA," *Courant Inst. Math. Sci. New York*, 2011.

[32] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, and P. Ivanov, "PyCUDA: GPU Run-Time code generation for High," *Perform. Comput.*, 2009.

[33] A. Klöckner, "PyCUDA Documentation," 2009.

[34] J. Shetty and J. Adibi, "The Enron email dataset database schema and brief statistical report," *Sci. Inst. Tech. report, Univ.*, 2004.