Optimization of Plagiarism Detection using Vector Space Model on CUDA Architecture

Preprint · September 2022

CITATIONS
0

READS
7

4 authors, including:

Akmal Jahan MAC
South Eastern University of Sri Lanka
22 PUBLICATIONS **77** CITATIONS
SEE PROFILE

Hasindu Gamaarachchi
Garvan Institute of Medical Research
40 PUBLICATIONS **287** CITATIONS
SEE PROFILE

Roshan Ragel
University of Peradeniya
190 PUBLICATIONS **1,035** CITATIONS
SEE PROFILE

Optimization of Plagiarism Detection using Vector Space Model on CUDA Architecture

Jiffriya Mohamed-Abdul-Cader[a], Akmal-Jahan Mohamed-Abdul-Cader[b], Hasindu Gamaarachchi[c] and Roshan G. Ragel[d]

[a]*Department of Information Technology, Sri Lanka Institute of Advanced Technological Education, Sammanthurai, Sri Lanka*

Email: macjiffriya@gmail.com

[b]*Faculty of Applied Sciences, South Eastern University of Sri Lanka.*

Email: akmaljahan@fas.seu.ac.lk

[c]*School of Computer Science and Engineering, University of New South Wales, Australia.*

Email: hasindu2008@gmail.com

[d]*Faculty of Engineering, University of Peradeniya, Sri Lanka.*

Email: roshanr@pdn.ac.lk

Biographical notes:

Ms. Jiffriya Mohamed Abdul Cader received her M.Phil degree in Computer Science from university of Peradeniya, Sri Lanka. She is a lecturer at the Department of Information Technology, Sri Lanka Institute of Advanced Technological Education Sammanthurai, Sri Lanka. Her areas of research interest include in text analysis, GPU computing and optimisation.

Dr. Akmal Jahan Mohamed Abdul Cader received her Ph.D in Computer Science from Queensland University of Technology, Australia. She is a lecturer at the Department of Computer Science, South Eastern University of Sri Lanka. Her areas of research interest include Pattern recognition, computer vision, image processing, document image analysis, machine learning, data mining and GPU computing.

Dr. Hasindu Gamaarachchi received his PhD in Computer Science and Engineering from UNSW Sydney, Australia. He is currently serving as a researcher at the Garvan Institute of Medical Research, Sydney and a conjoint lecturer at the School of Computer Science and Engineering, UNSW Sydney. His research interests are in embedded systems, GPGPU computing and bioinformatics.

Dr R.G. Ragel received his PhD in Computer Science and Engineering the University of New South Wales, Sydney, Australia. He is a Professor at the Department of Computer Engineering at the University of Peradeniya, Sri Lanka. He has co-authored more than 150 peer-reviewed articles in topics including Micro-Architectural Support for Reliability and Security in Embedded Processors (SoC), Internet of Things (IoT), Side Channel Attacks and Countermeasures, Application Specific Processor Design, High Performance Computing such as Hardware/Software Acceleration of Bioinformatics Algorithms and Wearable Computing.

Optimization of Plagiarism Detection using Vector Space Model on CUDA Architecture

Abstract

Plagiarism is a rapidly rising issue among students that occurs during submission of assignments, reports and publications in universities and educational institutions because of the easy accessibility of abundant e-resources on the Internet. To mitigate plagiarism among students, many tools are available for natural language plagiarism detection. However, they become inefficient when dealing with prolific number of documents with large content due to the time they consume. Therefore, we have proposed a way for software-based acceleration on text-based plagiarism detection using a suitable model on CPU/GPU. For the evaluation on CPU, initially a software-based serial vector space model was implemented on CPU and tested with 1000 text-based documents particularly, students' assignments, where it consumed 1641s for plagiarism detection. As computation time of the plagiarism detection is a bottleneck of performance while treating a prolific number of text-based sources with different sizes, we focus on accelerating and optimizing the model with the number of documents. Therefore, this research intends to implement and optimize the vector space model on the Graphics Processing Units (GPU) using Compute Unified Device Architecture (CUDA). In order to speed-up, a parallel version of the model was developed on GPU using CUDA, tested with the same dataset which consumed only 36s and gained 45x speed up compared to CPU, and when optimized further it took only 4s for the same dataset which was 389x faster than serial implementation.

keywords

Graphics Processing Units (GPU), Computer Unified Device Architecture (CUDA), Plagiarism Detection, Vector Space Model

1. Introduction

Plagiarism can be defined as a violation of the copyright of an author's or authors' literature work. It refers to copying others' work or stealing others' ideas as one's work without any proper acknowledgement which leads to diminishing quality of the work. Plagiarism is a rapidly rising issue among students which occurs during submission of assignments, reports, thesis and publications in universities due to easy accessibility of abundant electronic resources on the Internet. In a survey conducted in the University of California at Berkley, the percentage of plagiarism was increased by 74.4% within four years (Lukashenko, 2007). Another study carried out by Butakov and Scherbinin (Butakov, 2009) showed that most of the students from high school involved in this unprincipled activity. A trend of plagiarized assignment submission among students was analysed in the University of Bostwana which depicts that an average of 20.5% students involved in this immoral activity (Batane, 2010). Thus, the issue is a trend which needs to be controlled and minimized among students.

As the impact of plagiarism is rising, scrutinizing of one's literary work is crucial for evaluating it properly and equitably. Moreover, manual plagiarism detection needs great effort to examine originality of an assignment or a report. Hence, examiners would have to consume a massive amount of time in reviewing to provide a high-quality judgment on these documents manually. Further, manual detection may be impossible when source and suspicious documents are in an increased level. Therefore, an automated tool for plagiarism detection is vital to detect plagiarism

and to reduce the workload of the examiner while evaluating students' creative writing properly without wasting their valuable time.

Though a variety of automated tools for plagiarism detection exists, they are ineffective to deal with a large number of documents. Because the greater the number of documents available, the more comparisons to be handled. Moreover, these tools can manage limited number and size of documents, which consume massive amount of processing time for analysing the documents and it becomes unproductive process. For instance, when we deal with 200 documents, there are 19900 comparisons among them (200×199/2). If we assume that a tool consumes a second to check a pair of documents, it would take more than five hours (19900 seconds) to compare unique paired of documents. Since there are more comparisons and cross-checking among the whole document set, it is a time-consuming process. Moreover, some tools consume a significant amount of time to generate a report of the detection while handling enormous documents. In addition, these tools can treat only a limited number of documents and small in size at a time. As a result, the existing detection tools are incapable to detect plagiarism when treating a prolific number of documents. As detection time is a significant factor, it is crucial to optimize plagiarism detection by minimizing time consumption in this scenario. Therefore, this research intends to optimize plagiarism detection on a massive number of text-based documents efficiently and mitigate the plagiarism issue in an academic environment.

Generally, performance measurement of a tool is evaluated based on two metrics: i) speed and ii) accuracy. In our previous study (Jiffriya, 2013), we focused on speed of plagiarism detection, where AntiPlag tool was introduced using trigram sequence matching algorithm with a set of assignments, and it was compared with Plagiarism Checker X, which consumed more than two hours. This depicts that plagiarism detection is a time-consuming process. Moreover, time consumption for the detection increases dramatically with respect to the number of documents as there are searching processes and higher number of comparisons among them. Then we focus on accuracy of plagiarism detection, where a set of assignments were analysed using trigram sequence matching algorithm, and the accuracy was compared with the plagiarism detection accuracy of vector space model (Jiffriya, 2014). From this study, the vector space model showed more reliable results compared to the trigram sequence matching algorithm. Therefore, finding accuracy of plagiarism detection is out of the scope of this work and it focuses on speed of the detection for the text-based documents.

The objective of the research is to find a new effective approach to optimize plagiarism detection by reducing processing time. The contribution of this work is that we have investigated a new approach to detect plagiarism using vector space model to handle a large number of text-based documents and reduced the time taken for plagiarism detection from hours to minutes by accelerating the process using GPU, and the optimization is achieved through parallelism.

As plagiarism detection is a time-consuming process, the major contribution of the research is to propose an approach to enhance the efficiency of plagiarism detection when we deal with huge number of digital documents, which helps to easily identify plagiarized documents without wasting more time. Initially we implemented serial version of vector space model for plagiarism detection and tested with 1000 documents where it has 499,500 comparisons. Then the model has been developed in GPU using CUDA architecture in order to accelerate the detection process, where we have used five kernels for it, and tested with the same dataset. The second contribution of the work is the optimization of the process, where we focused on optimizing the kernels which consume greater time according to Amdhal's law. The optimized version was tested with the

datasets and showed 389x faster than the serial version. Eventually, we analysed profile of the model which depicts high occupancy rate and device utilization. Although plagiarisms detection process is a time-consuming process, the optimized version of vector space model reduces the time consumption significantly. Therefore, the model facilitates to detect plagiarism on large number of documents efficiently.

The remaining of the paper is structured as follows: Section 2 reviews related work of the existing plagiarism detection domain; Section 3 depicts the methodology where process of data collection and parallel implementation of plagiarism algorithm on GPU are included and the results are represented in Section 4 and Section 5 concludes findings of the research.

2. Literature Review

2.1 Plagiarism Detection Tools

Plagiarism can be avoided by two major approaches: prevention and detection. The former focuses on generating digital documents without making copy and distribution with the help of encryption techniques (Yu, 2017). The latter deals with identifying plagiarized portion and proportion of the digital documents. We focus on plagiarism detection approach for which several detection tools are applied for it.

The existing automated tools for natural language plagiarism detection can be categorized into intra-corpal and extra-corpal tools according to the area of the detection. Intra-corpal tools such as CopyFind (B.L) and CopyCatch (White, 2004) detect plagiarism within a set of grouped documents in a learning community, whereas EduTie (Lancaster, 2005) and EVE2 (Lancaster, 2005) are extra-corpal tools which check the similarity among available external sources of content on intranet and internet (Kats, 2010). Further, some tools such as iParadigms (Lancaster, 2005), WORDCheck (Clough, 2000) and Gossip (Lancaster, 2005) are capable for handling both intra and extra-corpal plagiarism detection.

On the other hand, the existing tools such as PlagAware (Malthan, 2006), PlagScan (Markus, 2014), iThenticate (Cross, 2017), CheckForPlagiarism.net (Academic Paradigms, 2004) and plagiarismdetection.org (Plagiarism Detection.org, 2008) are web-based commercial tools which are commonly used by institutions and students. A comparison among these tools based on features and performance has shown that PlagAware and iThenticate are in first consecutive places (Ali, 2011). However, they have some limitations till such as i) they can treat limited number of documents and certain file size at a time and ii) they utilize more time when checking a large number of documents. For an instance, PlagAware is unable to handle a document that is more than 15 MB in size and the execution time for the detection depends on the workload of servers and documents' size (Markus. 2014). Further, Turnitin is a web-based well-known plagiarism detection tool used by 35000 educational institutes throughout the world. Though it can support 400 pages with maximum size of 40 MB, students can check only a maximum of 0.5 MB document size within 150,000-character limitation, and the originality report may be generated within a minute to hours (Caren, 1998). Also, iThenticate and Ferret do not allow to check a document which is more than 25000 and 10000 words respectively (Ali, 2011; Lyon, 2004). Moreover, plagiarismDetection.org cannot support multiple document comparisons and consumes more time to display the analysed results (Ali, 2011). As a result, plagiarism detection tools become inefficient while large amount of text-data are processed due to the time consumption, and some tools are incapable for handling these number of documents. Thus, it is a significant bottleneck of these tools to generate the result while dealing with numerous sources and suspicious documents.

Therefore, this research indents to investigate an approach to handle the scenario efficiently.

2.2 Plagiarism Detection Approaches

In analysing plagiarism detection techniques used in the automated tools, there are two main approaches practiced to detect plagiarism between documents: a) content-based approach where documents are analysed based on the logical structure; b) stylometry-based approach where documents are investigated according to an author's style with an assumption that every author has a unique style (Tschuggnall, 2013). Stylometry-based approach is ineffective for analysing students' creative writing. Because it would be incredible to recognize writing style of students as they submit assignment for the first time. On the other hand, content-based approach which focuses on semantic features of a document, are very commonly used. A variety of algorithms were used for the extraction of semantic features of documents such as Winnowing, Fingerprinting (Foltýnek, 2020), Hashing (Shin, 2004), Heuristic (Menai, 2012; Foltýnek, 2020), deep neural network algorithms such as convolutional neural network and a recurrent neural network (Agarwal, 2018), and they gave a varying degree of precision although these algorithms are mostly used in video and image processing (Zeng, 2018). AlSallal et al. (AlSallal, 2019) applied integrated approach by combining semantic and stylometry approach to improve accuracy of plagiarism detection. However, these algorithms have some limitations when extracting features from the content and computing similarity such as i) processing time is a limiting factor as number of documents are increased and ii) efficiency of the detection declines with the size of documents. Therefore, there is a necessity to select an algorithm for accelerating plagiarism detection which should have ample number of independent tasks.

Table 1: List of tools and techniques that have been used for plagiarism detection and their limitations.

Author	Tools & Techniques	Limitations & Remarks
(Malthan, 2006; Markus, 2014)	PlagAware	Web-based commercial tool; Limited to the file size of 15 MB; Execution time for the detection depends on the workload of servers and documents' size.
(Markus, 2014)	PlagScan	Web-based commercial tool
(Cross, 1998)	iThenticate	Web-based commercial tool; do not allow to check a document which is more than 25000 words.
(Academic Paradigms, 2004)	CheckForPlagiarism.net	Web-based commercial tool
(Plagiarism Detection.org, 2008)	plagiarismdetection.org	Web-based commercial tool
(Caren, 1998)	Turnitin	Web-based commercial tool Support 400 pages with maximum size of 40 MB, the student can check only maximum 0.5MB document size with 150,000-character limitation and the originality report may be generated within a minute to hours.
(Ali, 2011; Lyon, 2004)	Ferret	Does not allow to check a document more than10000 words.
(Ali, 2011)	plagiarismDetection.org	Does not support multiple document comparisons and consumes more time to display the analyzed results.

Vector space model is well-known for information retrieval in data mining due to its better accuracy rate (Gudivada, 1997; Shin, 2004; Turney, 2010). For instance, the work in (Shin, 2004) used this model to retrieve related required documents from a medial library, and it shows a better performance. Since this model gives a higher precision, researchers focus on applying it to the other related fields of study such as semantic similarity (Reinhard, 2003) and plagiarism detection

(Rehurek, 2008; Zechner, 2009; Wang, 2013).

2.3 Vector Space Model

Documents are represented as a vector in a vector space model where terms in a document are weighted. Thus, the inverse document frequency shown in Equation (1) is multiplied with term frequency and form weighted term frequency vector using Equation (2). By calculating document length using Equation (3) eventually, the similarity score is calculated using cosine similarity measurement as depicted in Equation (4).

$$idf_t = log\ (N / df_t) \quad (1)$$

df_t - Document frequency for term t
idf_t - Inverse document frequency for term t
N - Total number of documents

$$Wtf_{d,t} = Ltf_{d,t} \times idf_t \quad (2)$$

$Wtf_{d,t}$ - weighted term frequency vector
$Ltf_{d,t}$ - Frequency of a term in a document
idf_t - Inverse document frequency for term t

$$\text{Doc Length } (Dl_d) = (\sqrt{\Sigma Wtf_{d,t}^2}) \quad (3)$$

$$Cos\ (D_i, D_j) = \Sigma\ (Wtf_{i,t} \times Wtf_{j,t}) / (\sqrt{\Sigma Dl_i^2} \times \sqrt{\Sigma Dl_j^2}) \quad (4)$$

$D_i D_j$ -Document Pair

As the model shows a better accuracy rate in our previous study, we plan to apply the existing model for plagiarism detection and accelerate it. Because, the model is effective while handling with a smaller number of documents with less content, but it takes much processing time to generate output as it deals with a huge number of documents due to the involvement of lots of string related computation among the combination pair. Also, the time consumption for the detection increases quadratically with respect to the number of documents. Therefore, it becomes a bottleneck and challenging issue when applying this model to a large number of documents due to the higher time consumption. On the other hand, CPU utilization was analysed on duplicate document detection using the model, which depicted 58% of CPU allocated for it (Yuan, 2011). Further, the speed of the detection reduces with an increased number of documents due to proceed the process sequentially by CPU. Therefore, we need to focus on acceleration and optimizing the model with the prolific number of documents. There are several advantages of vector space model which has been practiced in many applications, and none of the work has reported its parallel nature or how it can be used to accelerate the detection process. Therefore, this work attempts to investigate the nature of this model in the process of acceleration of detection.

2.4 Acceleration of Plagiarism Detection

Plagiarism detection techniques can be accelerated by hardware and software-based approaches. In software-based acceleration, two strategies are very frequent. They are: a) clustering approach where documents are categorized according to the similarity, and related documents are deviated from non-related documents (Li, 2018a; Li, 2018b); b) parallel computing that allows processing multiple tasks simultaneously. The parallel computing strategy is an effective technique to accelerate plagiarism detection when dealing with a large amount of data. Multi-core CPU and Graphics Processing Units (GPUs) can be used for software-based acceleration while Field Programmable Gate Arrays (FPGA) can be used for hardware-based acceleration (Grozea, 2010).

In our previous study (Jiffriya, 2013), we have investigated the followings: i) comparison of time using trigram sequence matching with existing tools; ii) comparison of plagiarism detection

accuracy using trigram sequencing and vector space model (Jiffriya, 2014) where vector space model yielded better accuracy and iii) investigation of speed up of trigram sequence matching algorithm on Compute Unified Device Architecture (CUDA) architecture using two datasets (with the size of 12.1MB and 20.3MB consisted 200 documents in each dataset), which gained 5 and 6 times speed up respectively with respect to CPU (Jiffriya, 2015). From the analysis, we have chosen vector space model to accelerate on CUDA architecture as it performed better in terms of accuracy. Since the objective of the work is to investigate how much vector space model can be accelerated on GPU using a set of text-based documents, particularly students' assignments, dependency of different type of documents or computation of detection accuracy are out of the focus in this work.

2.5 Compute Unified Device Architecture (CUDA)

CUDA is a parallel computing and programming model introduced by NVIDIA, which provides a cost-effective architecture to perform general purpose computations in a parallel way (Uribe-Paredes, 2011), and assists to develop a highly-demanding complex computation programmes in an efficient way by exploiting GPU (Menai, 2012). The programme should have ample of parallel processing and support for data level parallelism. GPU is a commonly used acceleration technique, and it facilitates to perform general purpose computations in a parallel way (Uribe-Paredes, 2011). CUDA C is a programming language which enables to communicate with GPUs and facilitates to execute kernels where multiple threads can be run simultaneously. GPU is used for accelerating many applications such as similarity detection (Cruz, 2015), image processing (Matam, 2011; Tsai, 2015) and computer vision (Wang, 2013). Sanjay P. Ahuj (Ahuja, 2020) used GPU to analyse the performance of two public cloud providers such as Amazon Web Services and the Google Cloud Platform using MPI architecture while Shadi AlZu'bi et al. (AlZu'bi, 2019) used GPU to increase efficiency of 3D image segmentation. As GPU is a cost-effective method for parallel computing in recent years, we have chosen GPU to accelerate the detection using CUDA architecture.

There are some similarity detection related algorithms applied on GPU for acceleration with different methodologies. For instance, a study was performed to accelerate similarity detection between documents using MinHash algorithm on GPU which gained 25 times speed up compared to serial implementation (Cruz, 2015). In another work, Simhash was developed on CUDA architecture to detect near duplicate documents which showed 18x time faster execution than on CPU (Feng, 2015). In addition, the optimized Shingle algorithm was developed for duplicate detection on CPU using SIMD technology and GPU using OpenCL, which gained 38.5% and 170% performance respectively (Yuan, 2011). However, to the best of our knowledge, none of them accelerates the vector space model on high-performance graphics cards. Therefore, we apply the model on GPU to increase the performance during the process of large number of documents. From the overall analysis, vector space model is good enough for similarity detection, and there are several independent tasks involved in the model, which help to handle in a parallel way. Therefore, we have implemented the model with CUDA on GPU and details of the implementation is explained in Section III.

3. METHODS and MATERIALS

3.1 Data Collection and Data Preparation

The dataset used for this experiment was collected from a secondary source from a public database PAN with different sizes in a text file format (Martin, 2017). Thousand text files were collected

where each text file consists of a set of words, diagrams and tables with a word range from 1000 to 50000. They were pre-processed to generate a list of words and the entire document size was about 87 MB. Then the documents were grouped into 20 divisible number from 20, 40, up to 1000 documents and each group was considered as a dataset, which formed 50 datasets with different size in order to test the performance.

3.2 Methodology

Figure 1 shows the overview of the implementation where data collected from the PAN website (Martin, 2017), is pre-processed and formed as global and local term lists which are then copied into device memory. Later, it undergoes few steps to generate a similarity score. The overview of the methodology is categorized into three phases: i) pre-processing; ii) list generation and iii) GPU implementation as illustrated in Figure 1.

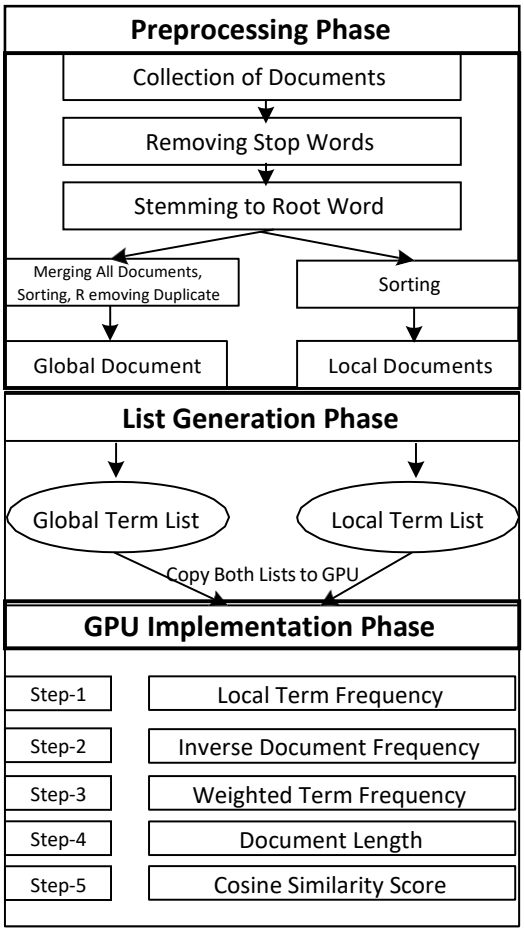

Figure 1. Overview of Parallel Implementation

3.2.1 Pre-processing

In the pre-processing phase, most of the frequent terms which are irrelevant for similarity detection known as stop words, have been removed from all documents and the remaining terms were stemmed to root form to extract important features of the documents. Terms in each document are sorted in an alphabetical order with duplicate terms and the output document is known as local document as illustrated in Figure 1. Meantime, all documents were merged into a single document and sorted its terms by removing duplicate terms, and the output document is named as global document. The global document consists of all unique terms of local documents. In the pre-

processing phase, the least impact terms for similarity detection have been removed and both kind of documents have a sorted list of terms which facilitates rapid search.

Initially, the identification of serial and parallel segments of the model is a crucial step to implement the vector space model on the CUDA architecture efficiently. Also, the algorithm should have enough parallelism to hide memory latency for the improvement of its performance. A CPU is capable for handling sequential computation efficiently while a GPU can carry out a huge number of parallel processes. Therefore, serial tasks have been allocated to CPU and simultaneous processes have been assigned to the GPU. The implementation is passed through: i) list generation and ii) GPU implementation phases.

3.2.2 List Generation Phase

In the list generation phase, the global and the local term lists have been formed by reading from global and local documents respectively. Then the both term lists have been copied from the host to the device memory. The detection process is a highly memory demanding task when dealing with enormous number of documents due to string related operations. Therefore, we have used dynamic memory allocation for utilizing memory efficiently.

3.2.3 GPU Implementation Phase

The GPU implementation is the last phase of the parallel implementation with CUDA architecture, where most of the computations of the model were processed. Though CUDA device has global, shared and local memories with divergent features, global memory is used to store global and local term lists in a linear fashion as all threads in a grid are capable to access it and its large storage capability. In this phase, it consists of five processing steps and was assigned to different kernels as illustrated in the Figure 1. Here as each step depends on the yield of its previous step, it was impossible to launch all kernels simultaneously. But each of these steps are parallelizable individually because terms in a list were independent and details of the implementation are given as follows:

Step -1

The first step was computing Local Term Frequency which was allocated to kernel-1, where threads have been mapped with global term list as illustrated in Figure 2. It shows formation of Local Term Frequency parallelly for a document. In the figure, each thread denoted as t_x in kernel-1 accesses each global term Gt_x consecutively, where x is the index, compares with local term list using binary searching techniques and identifies whether the global term is located or not in the local term list. If it is located, it would go through neighbour local terms $Lt_{x,y}$ to count its frequency by sequential searching and form the Local Term Frequency vector which is denoted as $Ltf_{x,y}$ in the figure where x and y are document number and term index respectively. These searching methods support to locate local term in the list quickly and minimize the time consumption for searching with the reduction of number of iterations. A similar process was repeated for each document to generate Local Term Frequency vector.

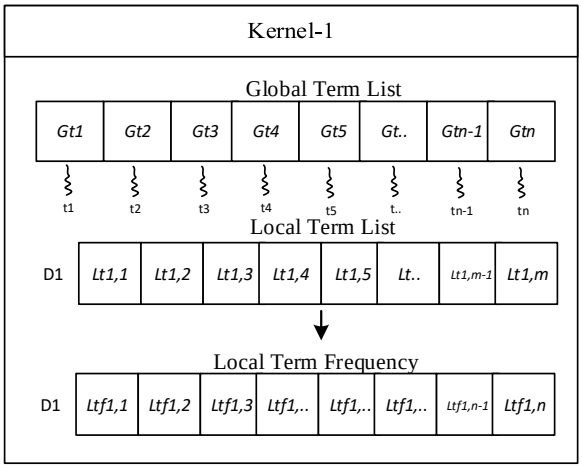

Figure 2: Structure of Kernel-1

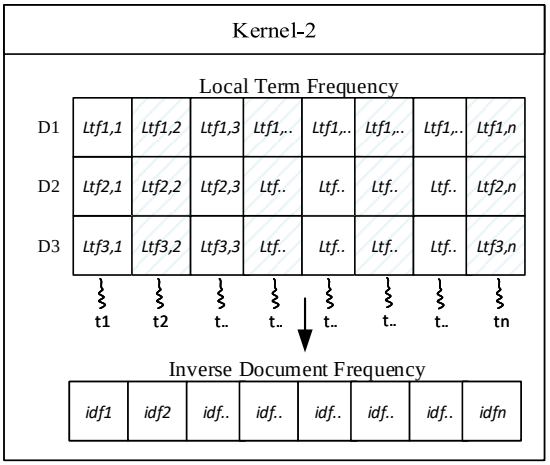

Figure 3. Structure of Kernel-2

Step -2

Calculation of the Inverse Document Term Frequency was assigned to kernel-2, where threads have been mapped to global term list as depicted in Figure 3. Each thread denoted by t_x accessed Local Term Frequency of the x index in all documents and counted the frequency of documents which had those terms, and calculated the inverse document frequency concurrently indicated by idf_x in the figure using Equation (1).

Step -3

The third step of the phase is used to compute Weighted Term Frequency where threads in kernel-3 have been mapped with the Local Term Frequency vector as shown in Figure 4. In the figure, each thread t_x accessed x index of Local Term Frequency and its inverse document frequency idf_x. Weighted term frequency was computed simultaneously by multiplying Local Term Frequency with its inverse document frequency as stated in Equation (2). It is represented as $Wtf_{x,y}$ in the Figure 4 where x and y indicate as document number and term index respectively, and the thread mapping part for it is explained in Section 3.3.

Step-4

Calculating document length of each document was allocated to kernel-4, where each thread t_x has been assigned to access weighted term frequency of every document $Wtf_{x,y}$ to compute its length concurrently using the Equation (3), which is denoted as Dl_x in the right side of Figure 5.

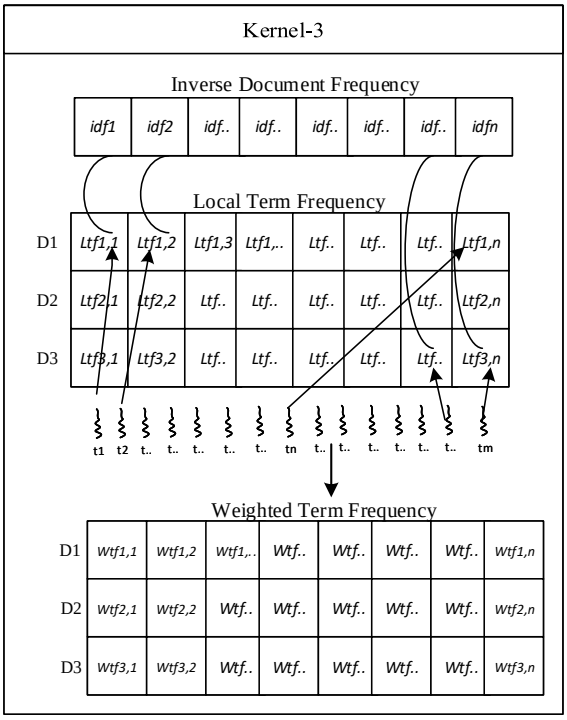

Figure 4: Structure of Kernel-3

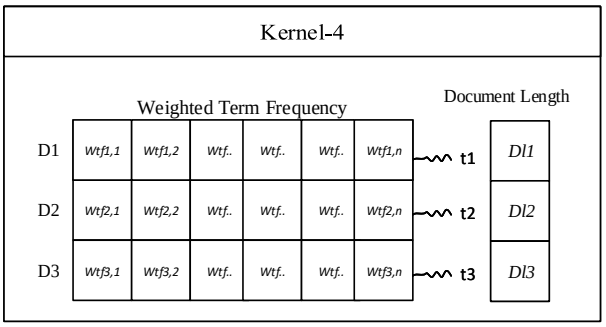

Figure 5: Structure of Kernel-4

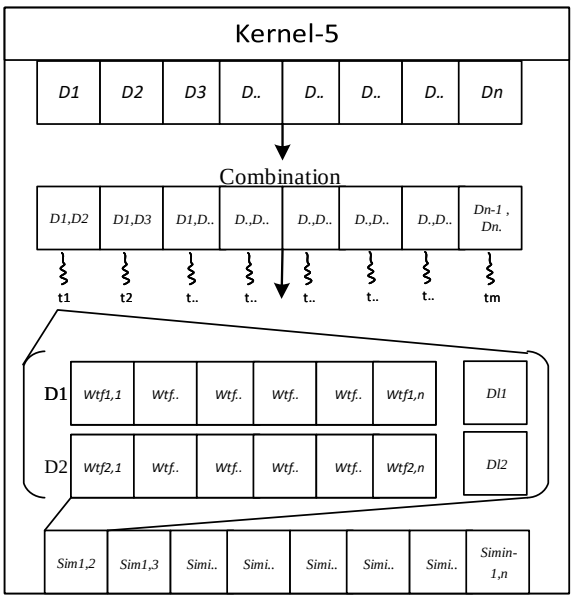

Figure 6. Structure of Kernel-5

Step-5

The final step is computing similarity score of each unique pairs of documents and it is assigned to kernel-5, where threads were mapped with each combination to compute similarity score between a document pair using the cosine similarity measurement simultaneously. In Figure 6, the first two rows show generation of combination among documents indicated as D_x, D_y where x and y are document numbers. Later portion depicts representation of documents in a combination as weighted term frequency $Wtf_{x,y}$ and document length Dl_x in the model. In the figure, each thread t_x accessed the following vectors: i) weighted term frequency and ii) document length of different paired documents. Then, the cosine similarity score was computed using Equation (4) indicated as D_i, D_j. The figure depicts the process of calculating the similarity score for a pair of documents D_1, D_2 by thread t_1. Similarly, all other threads run parallelly to compute the score for each pair of documents. In other words, the percentage of plagiarism of each pair of documents has been calculated simultaneously while running multiple threads which are equal to the number of combinations of document pairs. The combinations can be computed using Equation (5) where n is the number of documents in a dataset.

$$^{n}C_2 = n!/((n-2)!\times 2!) \quad (5)$$

3.3 The Selection of a Suitable Block Size

Thread is a small execution unit in CUDA and a collection of threads grouped together are known as a block. Number of threads in a block is called as block size. Selection of a suitable block size is one of the influential factors that affects the performance of an algorithm. In the parallel implementation, the selection of appropriate block size for each kernel is critical for which popular block sizes are chosen such as 256, 512, 768 and 1024, and the sizes are preferred to be a multiple of 32 for better performance as the warp size is 32 (NVIDIA, 2014). The maximum threads per block is 1024 in 3.5 computing capability. As the parallel programme is developed using five kernels, it is important to select an appropriate block size for each kernel for optimized performance. Hence, we selected a dataset with 800 documents as input and tested with the four mentioned block sizes such as 256, 512, 768 and 1024 in order to identify suitable block size for each kernel as block size is a critical factor of the performance. Time consumption of each kernel with the four block sizes were computed in seconds as illustrated in Table 2 where the least amount of time is considered as an appropriate block size of the kernel as shown in bolded.

Table 2: Consumption of Time of Kernels with Different Block Size

Block Size	Consumption of Time of Kernels (in Seconds)				
	Kernel-1	Kernel-2	Kernel-3	Kernel-4	Kernel-5
256	4.63372	0.00264	0.00558	**0.0787**	14.80296
512	**4.62444**	0.00262	0.00001	0.15597	15.72957
768	4.75439	**0.00243**	0.00001	0.23734	**12.44062**
1024	4.8563	0.00261	0.00001	0.27237	15.63989

According to the Table 2, the first two kernels such as kernel-1 and kernel-2 show better performance for the block size of 512 and 768 respectively than the other chosen sizes. In kernel-3, it does not show any deviation in execution time between block sizes except 256. Therefore, we have applied 2D thread indexing model for kernel-3 as illustrated in Figure 7. In the figure, each square denotes Local Term Frequency and document number. Each thread was assigned for per square to compute Weighted Term Frequency concurrently by accessing Local Term Frequency

and Inverse Document Frequency as depicted in the Figure 4. Block size (16, 16) for kernel-3 has shown better performance compared to other block sizes due to memory accessing pattern. In kernel-4, it shows an inverse relationship between block size and execution time. Since number of computations is less in the step, a few number of threads is enough to handle the process independently. Therefore, block size 256 was selected as the best size of block and less than this would not be sufficient for parallel computation. The final kernel named kernel-5 depicts better performance in 768 block size clearly compared to others.

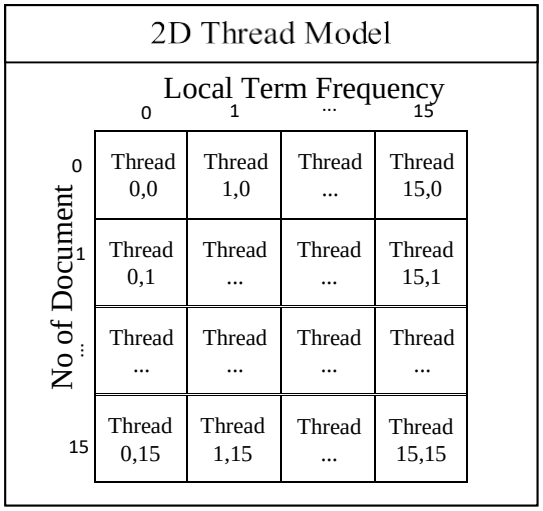

Figure 7. 2D Thread Model for Kernel-3

3.4 Optimization of Vector Space Model

In the process of optimization implementation, step-1 and step-5 were focused to optimize because these steps consume larger time compared to other steps as depicted in Figure 9. In *step-1* in *Section 3.2.3*, we have used a binary searching to locate local term and sequential search to compute its frequency. To optimize the step, the sequential search has been eliminated instead of that and local documents were generated with unique sorted terms and their frequency in the pre-processing phase. Hence, after locating local terms, it directly accessed frequency from the vector instead of sequentially searching the neighbouring terms. The pre-processing time on CPU only increases very slightly. In addition, it reduces the size of local documents by removing duplicate terms and enhances the process of locating local terms further with the reduction of iterations.

In *step-5*, computing the similarity score is assigned to kernel-5, where 1D thread indexing model was used and the indexing may be insufficient to run multiple thread simultaneously while the number of documents is increased. In addition to that, multiple threads try to access the same memory location simultaneously and it becomes a serial process which consumes a lot of time than the other kernels. Therefore, to optimize the step, we have applied a 2D thread model with the block size (16, 16). It is best suited to get better performance where it reduces the access to the same memory location simultaneously by using multiple threads at a time.

4. RESULTS AND DISCUSSION

4.1 Experimental Setup

For the experiment, a NVIDIA Tesla K40 graphics card, which has 3.5 CUDA compute capability with 2880 cores and 12 GB of memory was used. The card was installed on a computer with an Intel Xeon E5 2670 processor of 3.7 GHz and 32GB RAM. CUDA C language was used to implement the algorithm with CUDA 8.0 toolkit version. Initially, a serial version of the vector space algorithm was developed for plagiarism detection using C on CPU and tested using a pre-processed 50 datasets each of which comprises of 20, 40, 60 ... to 1000 text-based documents. The execution time for each dataset was recorded against the number of documents and size of documents. And then, the same algorithm was developed in a parallel approach with the help of GPU and CUDA platform using CUDA C, and the parallel version of the model was run on the Tesla K40 card with similar datasets. Execution time for both version on CPU and GPU plotted against the number and size of documents is shown in Figure 8. Time consumption for execution includes the following processes such as generation of lists, copying the lists and all computing steps into memory and writing the output to a file in both implementations, which were descriptively illustrated in the methodology. Both implementations were tested with different size of dataset on CPU and GPU to compare performance of the model.

4.2 Comparison of Performance

Figure 8 depicts execution time of serial and parallel version of vector space model on CPU and GPU respectively. According to the figure, at the beginning, there is no difference on performance between CPU and GPU due to lack of computation to hide memory latency. As the number of documents increases more than 300 with the size of 30MB, the time of the serial version on CPU rises dramatically, whereas the parallel version shows very slow increment with respect to the serial time.

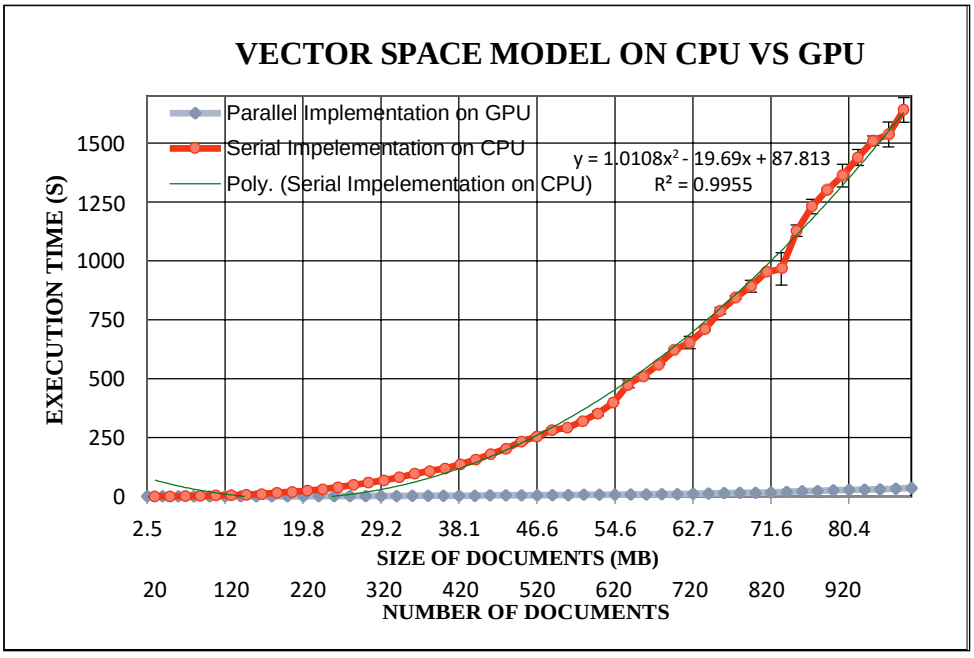

Figure 8: Performance Comparison of Vector Space Model on CPU and GPU

A curve fitted for serial implementation in Figure 8 shows the polynomial relationship between the execution time and documents' size on CPU. In order to show the relationship, Equation (6)

is generated with the highest goodness of fitness where x is the size of documents and y is the execution time. The execution time of vector space model on CPU increases polynomial with an increased number of documents, meanwhile execution time on GPU is nearly flat compared to CPU time.

$$y=1.01x^2-19.69x+87.81 \quad (6)$$

4.3 Time Consumption with Kernels

In an analysis of time consumption among kernels as shown in Figure 9, kernel-5 and kernel-1 are allocated for computing similarity score and Local Term Frequency. These kernels take approximately three quarters and a quarter portion of time respectively, whereas the other kernels consume negligible amount of time. Therefore, we can make an assumption according to the Amdhal's law that as if we apply optimization technique to kernel-1 and kernel-5, it will enhance the performance of the parallel implementation further. Therefore, we focused on optimization of the first and last step of the parallel implementation as they consume comparatively higher amount of time than the others as explained in Section 3.3.

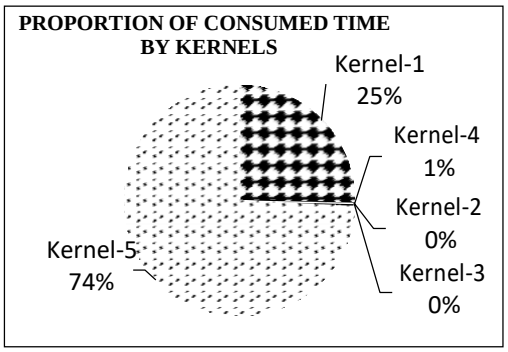

Figure 9: Proportion of Time Consumption

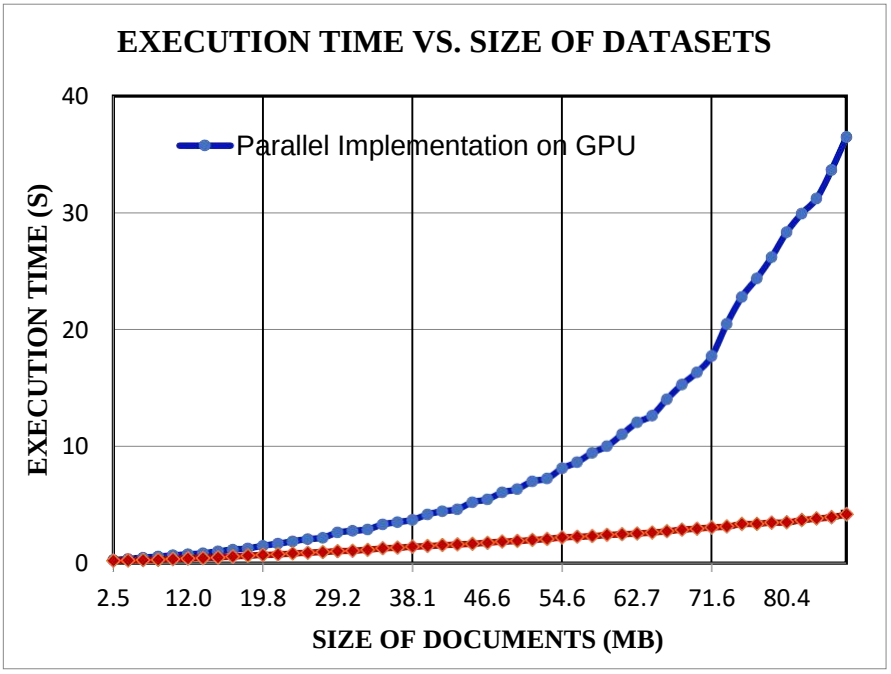

Figure 10: Optimization of Vector Space Model on GPU

4.4 Optimized Performance

Figure 10 shows performance of parallel and optimized parallel implementation on GPU. Initially, both performances are almost equal due to smaller size of dataset up to approximately 30 MB. But later, the execution time steadily increases from around 2s to 36s in parallel implementation while the time for optimized parallel implementation slowly rises from around 1s to less than 5s due to avoiding accessing of same memory location by multiple threads at a time.

Speedup can be defined as a rate of execution between different implementation. Table 3 shows the speedup of an optimized implementation on GPU compared to the serial implementation on CPU in terms of number of documents. Speedup is 9x when the number of documents is 100 while 1000 documents show a 389x speedup. According to the Table 3, it clearly depicts that speedup increases with the increased number of documents.

Table 3: Speedup of optimized implementation

Number of Documents	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
Optimized Parallel Implementation(s) on GPU	0.39	0.68	0.99	1.35	1.69	2.11	2.51	2.99	3.51	4.21
Serial Implementation (s) on CPU	3.65	19.66	58.19	118.72	233.16	352.41	622.02	892.33	1301.98	1641.41
Speedup	**9x**	**29x**	**59x**	**88x**	**138x**	**167x**	**247x**	**298x**	**371x**	**389x**

Table 4 depicts that the serial version of vector space model on CPU has taken 1641s to compute percentage of plagiarism between pairs of documents among 1000 documents with 499,500 pairs, whereas the parallel version on GPU has consumed only 36s for it. On the other hand, the optimized parallel implementation has required only 4s for the same dataset. Therefore, the parallel version shows 45 times faster than the serial and there is a 9x speedup between optimized and non-optimized parallel implementation on GPU. Further, the ratio between the time consumed for the serial version on CPU and optimized version on GPU shows 389x. In other way, optimized implementation is 389 times faster than the serial implementation. Execution time of parallel version increases exponentially with respect to the number of documents, whereas the time for the optimized version shows linear relationship and the increment is in minute.

Table 4: A Summary of Speedup on CPU and GPU

Serial Implementation on CPU	Parallel Implementation on GPU	Speedup
1641s	36s	45x
Parallel Implementation on GPU	Optimized Parallel implementation on GPU	Speedup
36s	4s	9x
Serial Implementation on CPU	Optimized Parallel implementation on GPU	Speedup
1641s	4s	389x

4.5 Profile of Optimized Vector Space Model

Table 5 depicts the overall profiling information of the optimized version of the model where optimization techniques were applied on kernel-1 and kernel-5 due to the greater time consumption for execution. In the table, the highest proportion of time is consumed by kernel-1, but before the optimization the highest proportion is in kernel-5 about 74% as shown in Figure 9, which is declined to 6% during optimization. In CUDA, the ratio between active warps and

maximum number of active warps supported by streaming multiprocessor is known as occupancy which depicts utilization of device in terms of computing units (Justin, 2011). Occupancy of kernel-1 is increased from 84% to 97% due to optimization. Further, except kernel-4 and kernel-5, all kernels have achieved greater occupancy rate with more than 96%, and these kernels utilize the device effectively due to hiding memory latency by computations. However, in kernel-4 memory latency cannot be hidden due to less amount of computation and unavoidable serial process. Though kernel-5 shows an increment in occupancy during optimization, it is still in lower level due to the process of loading and storing element into the memory.

Table 5: Profile of Optimized Vector Space Model on GPU

Profile Information of Optimized Parallel Based Implementation					
Properties	**Kernel-1**	**Kernel-2**	**Kernel-3**	**Kernel-4**	**Kernel-5**
Consumed Time Proportion	84.90%	0.10%	1.40%	7.60%	6.00%
Theoretical Occupancy	75.00%	75.00%	62.50%	100.00%	75.00%
Achieved Occupancy	72.70%	72.30%	61.30%	9.90%	10.20%
Achieved Occupancy (100%)	96.93%	96.40%	98.08%	9.90%	13.60%
Register Usage	24	12	41	18	33

5. CONCLUSION

This paper presents a parallel implementation of the vector space model for plagiarism detection using CUDA architecture on GPU which shows 45x speedup compared to serial implementation on CPU. The parallel version is optimized further on GPU according to the Amdhal's law, which gains 389 times speed up than the serial version of the vector space model on CPU. The contribution of the study shows a higher speed up compared to previous works on similarity detection. Even though, vector space model has string related operations and higher memory demand, we could achieve more than 96% occupancy in the first three kernels. Also, we believe that we will observe better performance further if we use latest Tesla K80 cards with 3.7 compute capability. In this study, we could not ensure reliability of the dataset because it has been collected from the external source. Also, there is some issue in collecting large number of documents primarily as it has some reputation issues, which affect standard and quality of institutions. Further, there is a low indication to achieve high rate of occupancy in the last two kernels. The future work of the study will focus on detecting plagiarism on reports and short-books and other publications. We will analyse performance of plagiarism detection tools on different types of documents such as assignments, reports, thesis, short-books and other documents to compare their dependency on different type of documents.

References

[1] Academic Paradigms, L., 2004. checkforplagiarism.net. [Online] Available at: https://www.checkforplagiarism.net [Accessed 20 April 2017].

[2] Ali, A., Abdulla, H., Snasel, V., 2011. Overview and comparison of plagiarism detection tools., Conference: Proceedings of the Dateso 2011: Annual International Workshop on Databases. pp. 161–172.

[3]B, L,. The plagiarism resource site. URL: http://plagiarism.bloomfieldmedia.com/wordpress/software/copyfind/.

[4] Butakov, S., Scherbinin, V., 2009. The toolbox for local and global plagiarism detection. Computers & Education 52, 781– 788. URL: http://www.sciencedirect.com/science/article/pii/S0360131508001930, doi: https://doi.org/10.1016/j.compedu.2008.12.001.

[5] Cross, C., 1998. *ithenticate*. [Online] Available at: http://www.ithenticate.com [Accessed 22 April 2017].

[6] Caren, C., 1998. *Turnitin*. URL: http://turnitin.com.

[7] Clough, P., 2000. Plagiarism in natural and programming languages: an overview of current tools and technologies.

[8] Feng, X., Jin, H., Zheng, R., Zhu, L., 2015. Near-duplicate detection using GPU-based simhash scheme, 223–228 doi:10.1109/SMARTCOMP.2014.7043862.

[9] Grozea, C., Bankovic, Z., Laskov, P., 2010. FPGA vs. Multi-core CPUs vs. GPUs: Hands-On Experience with a Sorting Application. pp. 105–117. doi:10.1007/978-3-642-16233-6_12.

[10] Gudivada, V., Raghavan, V., Grosky, W., Kasanagottu, R., 1997. Information retrieval on the world wide web. Internet Computing, IEEE, 58 – 68. doi:10.1109/4236.623969.

[11] Jiffriya, M.A.C., Akmal-Jahan, M.A.C., Gamaarachchi, H., Ragel, R., 2015. Accelerating text-based plagiarism detection using GPUs, IEEE 10th International Conference on Industrial and Information Systems (ICIIS), pp. 395–400. doi:10.1109/ICIINFS.2015.7399044.

[12] Jiffriya, M.A.C., Jahan, M.A.C.A., Ragel, R.G., 2014. Plagiarism detection on electronic text-based assignments using vector space model, in: 7th International Conference on Information and Automation for Sustainability, pp. 1–5.

[13] Jiffriya, M.A.C., Jahan, M.A.C.A., Ragel, R.G., Deegalla, S., 2013. Antiplag: Plagiarism detection on electronic submissions of text-based assignments, in: 2013 IEEE 8th International Conference on Industrial and Information Systems, pp. 376–380.

[14] Justin, L. & Rennich, S., 2011. CUDA warps and occupancy. NVIDIA Corporation.

[15] Kats, Y., 2010. Learning management system technologies and software. solutions for online teaching: Tools and applications. New York, Information Science Reference Hershey. doi:10.4018/978-1-61520-853-1.

[16] Lancaster, T., Fintan, C., 2005. Classifications of plagiarism detection engines. Higher Education Academy Subject Network for Information & Computer Sciences. ITALICS 4. doi:10.11120/ital.2005.04020006.

[17] Lukashenko, R., Graudina, V., Grundspenkis, J., 2007. Computer based plagiarism detection methods and tools: An overview, in: Proceedings of the 2007 International Conference on Computer Systems and Technologies, Association for Computing Machinery, New York, NY, USA. URL: https://doi.org/10.1145/1330598.1330642, doi:10.1145/1330598.1330642.

[18] Lyon, C., Barrett, R., Malcolm, J., 2004. A theoretical basis to the automated detection of copying between texts, and its practical implementation in the ferret plagiarism and collusion detector.

[19] Reinhard R., Johannes G., 2003. Word sense discovery based on sense descriptor dissimilarity. Proceedings of the Ninth Machine Translation Summit, page 315--322.

[20] Malthan, D., 2006. *Plagaware* URL: https://www.plagaware.com/

[21] Markus, G. & Johannes, K., 2014. *Plagscan* URL: https://www.plagscan.com/.

[22] Martin, P., Benno, S., Paolo, R. & Efstathios, S., 2017. PAN. [Online] Available at: http://pan.webis.de/data.html [Accessed 11 January 2017].

[23] Matam, K.K., Kothapalli, K., 2011. GPU accelerated Lanczos algorithm with applications, in: 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications, pp. 71–76.

[24] Menai, M., 2012. Detection of plagiarism in arabic documents. International Journal of Information Technology and Computer Science 4, 80–89. doi:10.5815/ijitcs.2012.10.10.

[25] Naseem, R., Kurian, S., 2013. Extrinsic plagiarism detection in text combining vector space model and fuzzy semantic similarity scheme. International Journal of Advanced Computing, Engineering and Application (IJACEA), 2(6).

[26] NVIDIA, 2014. CUDA toolkit documentation. NVIDIA Developer.

[27] Plagiarism Detection.org,2008. Plagiarismdetection.org [Online] Available at: URL: http://www.plagiarismdetection.org. [Accessed 22 April 2017].

[28] Rehurek, R., 2008. Plagiarism detection through vector space models applied to a digital library, in: Proceedings of Recent Advances in Slavonic Natural Language Processing. (RASLAN).

[29] S. H. Cruz, M., Kozawa, Y., Amagasa, T., Kitagawa, H., 2015. GPU acceleration of set similarity joins. International Conference on Database and Expert Systems Applications. doi:10.1007/978-3-319-22849-5_26.

[30] Shin, K., Han, S.Y., Gelbukh, A., 2004. Balancing manual and automatic indexing for retrieval of paper abstracts, in: Sojka, P., Kope£ek, I., Pala, K. (Eds.), Text, Speech and Dialogue, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 203–210.

[31] Tsai, P., Hsu, Y., Chiu, C.T., Chu, T.T., 2015. Accelerating adaboost algorithm using GPU for multi-object recognition, pp. 738–741. doi:10.1109/ISCAS.2015.7168739.

[32] Tschuggnall, M., 2013. Specht, G., Detecting plagiarism in text documents through grammar-analysis of authors.

[33] Turney, P.D., Pantel, P., 2010. From frequency to meaning: Vector space models of semantics. Journal of Artificial. Intelligence. Research. 37, 141–188.

[34] Uribe-Paredes, R., Valero-Lara, P., Arias, E., Sanchez, J., Cazorla, D., 2011. Similarity search implementations for multi-core and manycore processors, pp. 656 – 663. doi:10.1109/HPCSim.2011.5999889.

[35] Wang, G., Xiong, Y., Yun, J., Cavallaro, J.R., 2013. Accelerating computer vision algorithms using OPENCL framework on the mobile GPU - a case study, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 2629–2633.

[36] White, D., Joy, M., 2004. Sentence-based natural language plagiarism detection. ACM Journal of Educational Resources in Computing 4, 1–20. doi:10.1145/1086339.1086341.

[37] Yuan, X., Long, J., Zhang, H., Zhang, Z., Gui, W., 2011. Optimizing a near-duplicate document detection system with simd technologies. Journal of Computational Information Systems 7, 3846–3853.

[38] Zechner, M., Muhr, M., Kern, R., Granitzer, M., 2009. External and intrinsic plagiarism detection using vector space models, in: In Steinet.

[39] Batane, T., 2010. Turning to Turnitin to Fight Plagiarism among University Students. Educational Technology & Society, 13(2), p. 1–12.

[40] Agarwal, B., Ramampiaro, H., Langseth, H. and Ruocco, M., 2018. A deep network model for paraphrase detection in short text messages. Information Processing & Management, 54(6), pp.922-937.

[41] AlSallal, M., Iqbal, R., Palade, V., Amin, S. and Chang, V., 2019. An integrated approach for intrinsic plagiarism detection. Future Generation Computer Systems, 96, pp.700-712.

[42] Foltýnek, T., Meuschke, N. and Gipp, B., 2020. Academic Plagiarism Detection. ACM Computing Surveys, 52(6), pp.1-42.

[43] Zeng, Z., Li, Z., Cheng, D., Zhang, H., Zhan, K. and Yang, Y., 2018. Two-Stream Multirate Recurrent Neural Network for Video-Based Pedestrian Reidentification. IEEE Transactions on Industrial Informatics, 14(7), pp.3179-3186.

[44] Yu, C., Li, J., Li, X., Ren, X. and Gupta, B., 2017. Four-image encryption scheme based on quaternion Fresnel transform, chaos and computer generated hologram. Multimedia Tools and Applications, 77(4), pp.4585-4608

[45] Ahuja, S., Czarnecki, E. and Willison, S., 2020. Multi-Factor Performance Comparison of Amazon Web Services Elastic Compute Cluster and Google Cloud Platform Compute Engine. International Journal of Cloud Applications and Computing, 10(3), pp.1-16.

[46] AlZu'bi, S., Shehab, M., Al-Ayyoub, M., Jararweh, Y. and Gupta, B., 2020. Parallel implementation for 3D medical volume fuzzy segmentation. Pattern Recognition Letters, 130, pp.312-318.

[47] Li, Z., Nie, F., Chang, X., Nie, L., Zhang, H. and Yang, Y., 2018. Rank-Constrained Spectral Clustering with Flexible Embedding. IEEE Transactions on Neural Networks and Learning Systems, 29(12), pp.6073-6082.

[48] Li, Z., Nie, F., Chang, X., Yang, Y., Zhang, C. and Sebe, N., 2018. Dynamic Affinity Graph Construction for Spectral Clustering Using Multiple Features. IEEE Transactions on Neural Networks and Learning Systems, 29(12), pp.6323-6332.