ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/304298204

Accelerating text-based plagiarism detection using GPUs

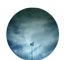
Conference Paper · December 2015

DOI: 10.1109/ICIINFS.2015.7399044

CITATIONS
5

READS
188

4 authors:



Jiffriya Mohamed Abdul Cader
Sri Lanka Institute of Advanced Technological Education
6 PUBLICATIONS **38** CITATIONS
SEE PROFILE

Akmal Jahan MAC
South Eastern University of Sri Lanka
20 PUBLICATIONS **76** CITATIONS
SEE PROFILE

Hasindu Gamaarachchi
Garvan Institute of Medical Research
38 PUBLICATIONS **284** CITATIONS
SEE PROFILE

Roshan Ragel
University of Peradeniya
188 PUBLICATIONS **1,016** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Hand Biometrics View project

PhD research of S. M. Vidanagamachchi under the supervision of myself (Dr. S. D. Dewasurendra), Dr. R. G. Ragel and Prof. M. Niranjan View project

All content following this page was uploaded by Roshan Ragel on 20 July 2016.

The user has requested enhancement of the downloaded file.

Accelerating Text-based Plagiarism Detection Using GPUs

MAC Jiffriya, MAC Akmal Jahan
Post Graduate Institute of Science
University of Peradeniya
Peradeniya, Sri Lanka

Hasindu Gamaarachchi, and Roshan G. Ragel
Department of Computer Engineering
University of Peradeniya
Peradeniya, Sri Lanka

Abstract— Plagiarism is known as an unauthorized use of other's contents in writing and ideas in thinking without proper acknowledgment. There are several tools implemented for text-based plagiarism detection using various methods and techniques. However, these tools become inefficient while handling a large number of datasets due to the process of plagiarism detection which comprises of a lot of computational tasks and large memory requirement. Therefore, when we deal with a large number of datasets, there should be a way to accelerate the process by applying acceleration techniques to optimize the plagiarism detection. In response to this, we have developed a parallel algorithm using Computer Unified Device Architecture (CUDA) and tested it on a Graphical Processing Unit (GPU) platform. An equivalent algorithm is run on CPU platform as well. From the comparison of the results, CPU shows better performance when the number and the size of the documents are small. Meantime, GPU is an effective and efficient platform when handling a large number of documents and high in data size due to the increase in the amount of parallelism. It was found out that for our dataset, the performance of the algorithm on the GPU platform is approximately 6x faster than CPU. Thus, introducing GPU based optimization algorithm to the plagiarism detection gives a real solution while handling a large number of data for inter-document plagiarism detection.

Keywords— CPU, GPU, NVIDIA, CUDA, Jaccard similarity, vector space model, hashing strategy, thread, block.

I. INTRODUCTION

The term 'plagiarism' refers to copying others' work or stealing others' idea as one's work without any acknowledgment. Plagiarism is a current growing issue not only in the academic field but also in creative activities such as literature, articles, poetry, songs, cinema and web content. Nowadays many of the scholars have faced the problem of dealing with plagiarized content, either while reviewing research articles or evaluating assignments. In one of the surveys conducted at the University of California at Berkley, they realized that the percentage of plagiarism has increased by 74.4 % within four years [1]. Another study by Butakov et al. showed that most of the high school students involve in this unethical activity [2]. The internet widely opens the door of learning resources to learners and surfers. Therefore, plagiarism is becoming a growing problem as the learning sources are widely spread out and easily available to the user using electronic materials. It is unethical behavior that diminishes the art of quality of one's work and leads to produce bad behavior of a moral society [8].

Nowadays, there are computerized tools available to detect plagiarized content. However, these tools and techniques have several limitations. They can handle a limited number of documents and limited file size at a time and consume much more time when checking a large number of documents and large size papers. Even though, these tools and techniques help academics to mitigate the issues of plagiarism; the time consumed for this process becomes a significant challenge when we deal with a massive number of documents with substantial content. As there should be more cross-checking among whole documents to eliminate plagiarized documents during the evaluation, it is a more challengeable part for the academics when dealing with an increased number of documents.

Plagiarism detection tools are inefficient when handling a massive amount of datasets. With an increasing number of documents, a higher number of comparisons and searching among them should be performed. Therefore, the time consumption of the plagiarism detection process is rapidly increased with respect to the size of the database or size of the documents. Meantime, CPUs handle all the tasks in a sequential manner. A research based on duplicate document detection found that plagiarism detection process occupied 58% of CPU usage [12]. For this reason, plagiarism detection process should be accelerated when handling a large amount of data.

Algorithm, hardware, and software based techniques can be used for acceleration of algorithm or application. Data clustering [13], [15] and parallel computing [14], [16] approaches are most common accelerating strategies of algorithms. Utilizing parallel approach is useful to speed up computation and reduces time consumption for the process. Software related parallel approaches can be implemented on multi-core CPUs and GPUs while hardware-based approach can be implemented on Field Programmable Gate Arrays (FPGA) [9]. Among them, Graphical Processing Unit (GPU) is a cost-effective architecture that provides powerful parallel computing facilities [9], [12], [17].

The existing research on the acceleration of similarity detection is based on either software or hardware based. In software approach, Parallel Failure-less Aho-Corasick Algorithm (PFAC) was developed for string matching in the application of network intrusion detection on GPU, which improved throughput and memory efficiency than Aho-Corasick algorithm on a CPU [23]. An FPGA is one of the parallel hardware implementation to speed up similarity detection, which used to check web attack in HTTP documents and achieved throughput 37 MB/s in real time data [24]. However, in the literature, a very few researchers address the acceleration of plagiarism detection either using

software or hardware, which motivated us to work on parallelization of computation when the data size is increased.

With an increasing number of documents, the number of combinations to be tested increases quadratically. The time consumption of a traditional CPU that does sequential execution would, therefore, increase rapidly with the number of files to be checked. With the rapid rise of the available documents, some acceleration is needed to perform plagiarism detection in a reasonable time. The aim of this work is to accelerate plagiarism detection on text-based documents by a parallelized implementation using a GPU. Thus, a trigram sequence matching algorithm was implemented on the GPU. In trigram technique, three words are considered at a time for performing the comparison between documents.

The rest of the paper is organized as follows: In Section II, we introduce the background, and in Section III, we present a literature survey. In Section IV, we present the methodology used. In Section V, we present our experimental results and a discussion of the results, and we conclude this paper in Section VI.

II. Background

A) *Trigram sequence matching algorithm*

To develop an effective plagiarism detection tool, different types of techniques can be used such as N-gram [15], vector space model [8], clustering and sampling methods [9], hashing strategies [10] and grammar analysis method [18]. A trigram is one of the widely used techniques for intrinsic plagiarism detection, and it is more suitable out of all N-gram techniques [15]. Therefore, we are focusing on the trigram sequence matching algorithm.

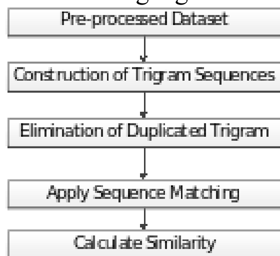


Fig. 1. The flow of the Algorithm

The basic steps of trigram sequence matching algorithm for plagiarism detection are depicted in Fig. 1. Stop words and insensitive words are removed in the pre-processing phase. Subsequently, all the documents in the dataset are converted to a pool of trigram sequences that consists of three consecutive words and are stored in an array-like data structure. Then the duplicate trigram sequences of each document are eliminated from the array. Next each document is identified using the index of the array and is compared with all other document using trigram sequence matching technique. Finally, a similarity score is computed by the use of Jaccard similarity score stated in Equation (1). In the equation, J is the Jaccard similarity, and d_i and d_j are the two documents.

$$J(d_i, d_j) = |d_i \cap d_j| / |d_i \cup d_j| \quad (1)$$

B) *Compute Unified Device Architecture (CUDA)*

Graphics Processing Unit (GPU) is a special hardware designed especially for graphics processing tasks such as 3D motion, image processing, object transformation and video processing [20]. To accelerate graphics processing tasks that are inherently parallel in nature GPUs are built with hundreds of cores and thousands of threads.

The concept called General Purpose Computing on Graphics Processing Units (GPGPU) which emerged some time back has paved the way to use GPUs for general purpose calculations that are traditionally done on CPU. Modern GPUs that have become extremely powerful with the advancement of graphics let parallel algorithms run extremely faster than on a traditional CPU. The performance of a parallel algorithm on a GPU can be several times faster than on a high-performance server, and hence GPU can be considered as a cost-effective solution to accelerate parallel applications [14], [19]. Compute Unified Device Architecture (CUDA) is the programming model, and the parallel computing platform introduced by NVIDIA [20]. It can be considered as an intermediate software layer that allows the programmer to use NVIDIA GPUs for general purpose computation. CUDA C is a programming language that has been formed by adding extensions to standard C language. It includes new syntax extensions that enable handling GPU specific tasks such as handling device memory and execution of threads. It also includes Application Programming Interface (API) functions that make the internal tasks such as memory allocation and memory copying convenient. In addition to general functions, a special type of function called CUDA kernels exists in CUDA C. A CUDA kernel that is generally written targeting the action of a single thread can be called with different thread configurations so that they would run in parallel on the GPU.

CUDA supports up to three-dimensional thread indexing. Multi-dimensional thread hierarchy lets the programmer to conveniently map components of the parallelized algorithm to GPU threads. For example, one-dimensional thread indexing would naturally map to a vector operation while two-dimensional thread indexing would naturally map to matrix operation.

In CUDA, a group of threads is called a thread block. Thread blocks for a particular CUDA kernel are equal in size, and the size must be specified by the user when calling the CUDA kernel. A block can have a maximum of 1024 threads. The size of a thread block is a significant factor that determines the performance of a CUDA application and, therefore, must be selected appropriately.

III. Literature Survey

There are two types of plagiarism detection tools widely used for plagiarism detection: i) source code; ii) text-based plagiarism detection. Plagio Guard, JPlag [3], Moss [4], Detecta Copius [5], Sherlock [6], Copy/Paste Detector (CPD) and Big Brother [7] are examples of popular tools for source code plagiarism detection. Text based plagiarism detection tools can be developed using one or more techniques either as standalone applications such as CopyCatch, WORDCheck, CopyFind [7] and Ferret or web-

based applications such as Turnitin, Article Checker and DupliChecker [11].

Developing any application using any technique should compare each document one by one and calculate a similarity score. All the above mentioned tools run on the CPU and hence with the increasing number of documents they consume a considerable amount of time.

The only work we found on accelerating plagiarism detection using GPU is presented in [12]. Plagiarism detection was used on funding project proposals using the Shingle algorithm, and it was optimized for GPU technology. The Shingle algorithm was developed using OpenCL on a GPU platform, and this implementation gains 170% of the performance compared to CPU architecture [12]. In our approach, we have applied trigram sequence matching algorithm for plagiarism detection, and the algorithm has been constructed with the use of CUDA on GPU platform.

IV. CUDA Implementation

the algorithm considered should have enough amount of parallel tasks to hide the memory latency and to improve the performance on the GPU. Assigning sequential tasks or tasks with only a small deal of parallelism onto the GPU would rather decrease the performance when compared to the CPU. Therefore, an important step of the CUDA implementation is the identification of sequential (or less parallel) and highly parallel portions of the algorithm. Then sequential (or less parallel) tasks are assigned to be executed on the CPU while highly parallel tasks are assigned onto the GPU. Subsection A below discusses how the serial (or less parallel) portion has been implemented on the CPU. Subsection B discusses how the parallel portions of the algorithm are implemented in a CUDA kernel. Implementing a CUDA kernel alone would not yield much performance improvement. Factors such as the thread block size and the memory access pattern highly affect the number of threads that run parallel. Subsections C and D discuss some techniques that we have followed to improve the performance on CUDA.

A) Serial or less parallel portion of the Algorithm

The initial process of the experiment is pre-processing of data. In this phase delimiters and stop words are eliminated from the documents. Then, file system operations such as the reading of files have to go through the CPU. Therefore, via the CPU documents were read one by one from the hard disk, constructed as a pool of tri-gram sequences and stored in a one-dimensional array. Documents were identified in the array using the index of the array by counting the number of trigram sequences of each document. Construction of trigram sequences is done in CPU, as the process does not have a high degree of parallelism.

Furthermore, duplicated trigram sequences of the documents are removed from the array, and the sequences were rearranged to reduce the memory wastage. The calculation of the number of combinations is a one-time task that can be directly calculated by the number of documents in the directory using the Equation (2). In the equation, ${}^{n}C_2$ is the number of combinations and n is the total number of documents. Hence, this is done on the CPU.

$$ {}^{n}C_2 = n!/((n\text{-}2)!\times 2!) \tag{2} $$

As a summary, representation of documents as a bag of trigram sequences, removing of duplicated sequences, and computing number of combinations are selected as sequential (or less parallel) steps to be executed in the CPU.

B) Parallel portion of the Algorithm

In the dataset, every document is independent of other documents. Trigram of a pair of documents are compared with all the other tri-gram sequences of another document and similarity score is computed using the Jaccard similarity measure as stated in Equation (1). Similarly, all pairs of documents are compared, and the similarity is calculated. This task is highly parallel especially when the number of documents is high. For example, n number of documents forms $n(n\text{-}1)/2$ numbers of combinations which produces that much of parallel tasks. Therefore, comparison of each pair of documents and computing similarity score are performed on GPU.

In our implementation, multiple threads are launched such that each thread handles a unique pair of documents. Each thread is assigned to each pair of the combination as shown in Fig. 2. Here Dx depicts a document where x is the document number. In the figure, n number of such documents is shown. Each document pairs with all other documents to form $n(n\text{-}1)/2$ number of combinations. Each combination has been assigned to a single thread. For example, thread 0 has been assigned to the document pair 1 and 2 while thread 1 is assigned for document pair 1 and 3, and so on. The assignment of document pairs to threads is graphically depicted in Fig. 2. There, the second row which has blocks numbered as D1, D2 and so on represents a document. There are n numbers of such documents. Then the third row shows how the pairwise combinations are formed using the documents. Then at the bottom how each combination is assigned to a thread is depicted.

The CUDA kernel that specifies what a single thread should execute was written such that it reads the pair of documents it is responsible, does the string matching and calculates the similarity score between the two documents. When multiple threads are launched in parallel, and each pair of the document are assigned to each thread, all threads call the string matching function and compute similarity score simultaneously for each combination.

C) Selection of the size of Block

For optimizing performance on GPU, we should increase the occupancy of the multiprocessor that is the ratio between a number of active warps and the maximum number of supported warps [21]. When the occupancy is increased, it hides memory latency that leads to improving the GPU performance [19]. The block size is one of the factors that affect the number of threads that run in parallel and hence the occupancy. To find out the appropriate number of threads per block, first the number of registers used per thread was determined with the help of the CUDA compiler. The

number of registers used by threads in our algorithm was 20. Then by inputting this value to the Occupancy Calculator provided by NVIDIA the proper block size was found out. We have selected 1024 threads as a suitable block size for the best performance.

D) Memory Allocation

From various types of memories in CUDA, we have used global memory to store the documents. As described in Subsection A, the documents were stored in a one-dimensional array in the RAM. This array was directly copied to the global memory of the GPU by using the respective CUDA API functions. Global memory is used for the following reasons: i) it is capable of accommodating a large number of documents as the memory size is large, and ii) all threads in any thread block have access to this memory. How the array resides in the global memory is depicted in Fig. 2. There the first row named as Doc1, Doc2 is a linear array where each slot represents a trigram.

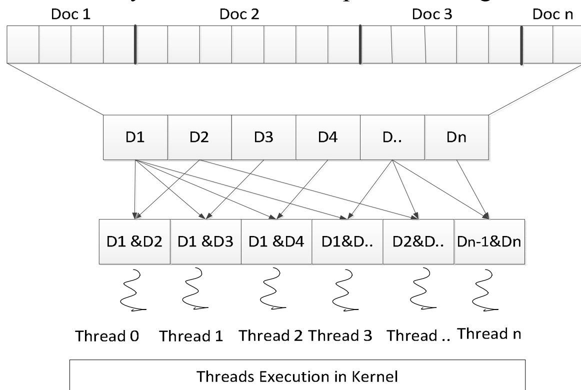


Fig. 2. Structure of the Kernel

The reason why the documents were stored in an array is necessary to emphasize. In CUDA, it is imperative to lay out data in the memory such that memory accesses coalesce. When multiple threads access different memory locations at the same time, all those memory accesses are coalesced such that all those memory locations can be fetched using one or few memory accesses. However, this is only possible if the memory locations accessed by those threads are lying contiguously or else the accesses would be serialized reducing the performance. In traditional programs for the CPU, a data structure like a linked list or graph representation would be suitable for representation of a document. However, using such a data structure with least spatial locality would cause scattered memory access instead of coalesced memory access that in turn would massively reduce the performance. Therefore, we have arranged all documents in a contiguous manner to facilitate contiguous access of memory.

In an application like plagiarism detection, the number of documents is known only at runtime. Therefore, the memory has to be allocated during runtime. Currently, API functions in CUDA for doing dynamic memory allocation only supports single dimensional arrays. That is the reason why we have stored the documents in a single dimensional array rather than a multi-dimensional array. In our work, each document was identified using the index of the array which is computed by counting the total number of trigram from each document. All the trigrams for a document in the array were accessed in a sequential manner.

E) The Overall Implementation

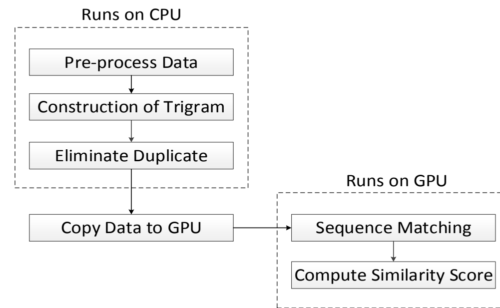


Fig. 3. Plagiarism Detection on CPU and GPU

Fig. 3 depicts the overall implementation of the plagiarism detection process on CPU and GPU. The scope of the algorithm is to calculate the similarity score and the execution time in both CPU and GPU platforms. The explanation of CPU implementation is detailed in Section II.

V. Results and Discussion

A) Experimental Setup

Dataset-1 and dataset-2 used here are taken from [22]. Two different sets of data, each of them has 200 documents in which dataset-2 is larger (documents having more text) than dataset-1. Further, in each dataset, all the documents were different in size.

The two pre-processed datasets (dataset-1 and dataset-2) have been tested using the developed algorithm on a Linux-based platform on a 64 bit i3 CPU. Similarly, the same datasets have been tested on GPU architecture using NVIDIA Tesla C2075 graphics card with 448 CUDA cores. Total execution time (time for all tasks shown in Fig. 1 except preprocessing) for the plagiarism detection was compared with respect to the number of documents and the size of documents on CPU and GPU platform.

B) Impact of the number of documents on the performance

Fig. 4 shows how the execution time for dataset-1 and dataset-2 on both platforms varies with the number of documents. Execution time on CPU has increased almost quadratically with the number of documents. The time on GPU has slowly increased and has become nearly constant when the number of documents is more than 100. When the number of document is less than 60, there is no performance gain by using the GPU. The reason is that less number of documents only launches a little number of threads that run in parallel which is inadequate to hide the memory latency in the GPU. Therefore, GPU architecture sometimes may consume more time than CPU when the number of files is minuscule. When the number of documents is more than 150, the number of combination is more than 10,000 which means that more than 10,000 threads execute

simultaneously. At such amount of parallelism, the memory latency is hidden. Therefore, GPU architecture is excellent when handling a large number of documents.

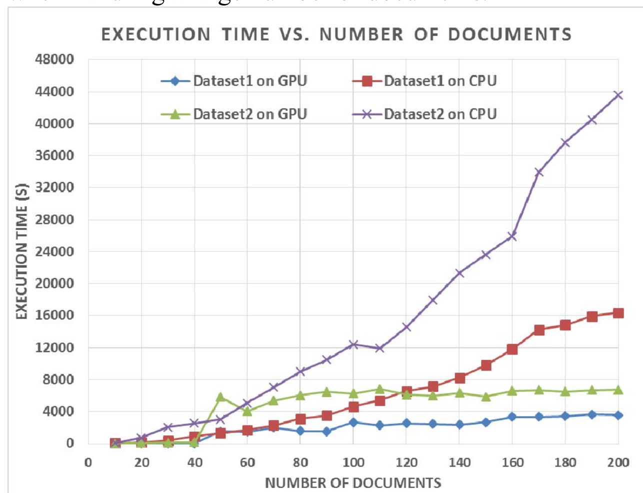


Fig. 4. Execution time of plagiarism detection on CPU and GPU

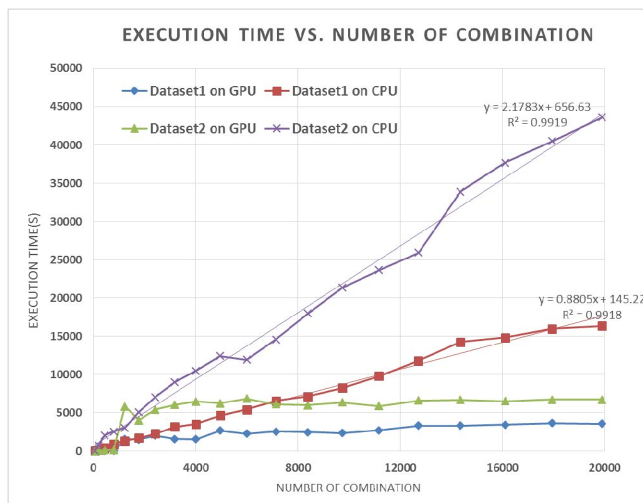


Fig. 5. Execution time of plagiarism detection based on number of combinations

Fig. 5 depicts the execution time on both CPU and GPU for dataset-1 and dataset-2 when the number of combinations is varying. In CPU, the time is linearly increased with the number of combinations. For the two CPU curves, we have curve-fitted two linear equations for dataset-1 and dataset-2 as in equation (3) and (4) with the highest R^2 value that shows the goodness of fitted model. In equation (3) and (4), y is the execution time and x is the number of combinations used for the plagiarism detection. Both show a linear relationship.

$$y = 2.1783x + 656.63 \quad (3)$$

$$y = 0.8805x + 145.22 \quad (4)$$

Meanwhile, the execution time on GPU becomes constant when the number of combinations is increased more than 10,000. The deviation of execution time between dataset-1 and dataset-2 is due to the size of each document. In our experiment, dataset-2 is larger in size compared to dataset-1. Thus, the size of the documents is another factor that has an influence on execution time.

C) Impact of the document size on the performance

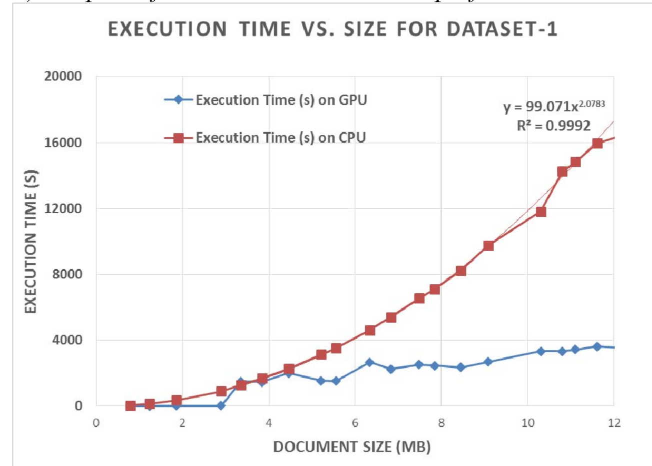


Fig. 6. Execution time based on size of the documents in dataset-1

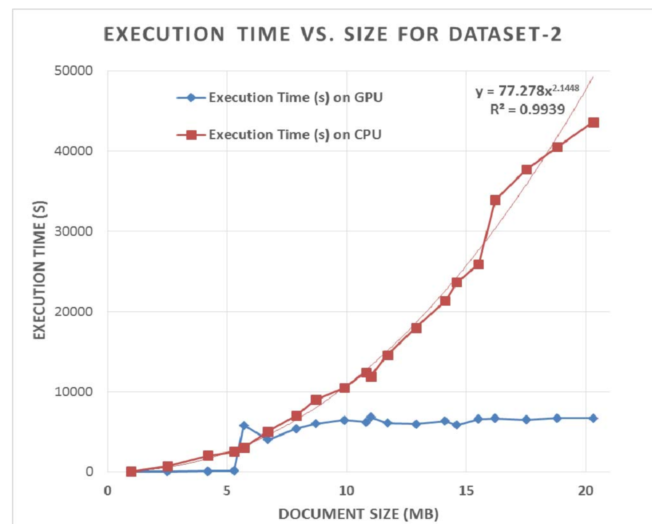


Fig. 7. Execution time based on size of the documents in dataset-2

We have compared execution time with the size of the documents in dataset-1 and dataset-2 as illustrated in Fig. 6 and Fig. 7. The time on CPU platform is increased quadratically with respect to the size of the document. Meantime, the time on GPU platform slightly increased up to 10MB in size (this is the size of the whole dataset) in dataset-1 and then remain almost constant in both datasets. Therefore, when the size of each document is increased the execution time is reduced on GPU architecture compared to CPU. In contrast to this, there is a rapid rise in the execution time of CPU architecture. Moreover, we can predict (based on Equation (5) and Equation (6)) that when we handle around 40-megabyte dataset, it will take approximately 58 hours for the execution of the algorithm on CPU while it takes only less than 2 hours on our GPU platform. This result indicates a speedup of about 30x. However, due to the significant time consumed by CPU for such experiment, we did not perform the same with such datasets.

The curve fitting results of the CPU time on Fig. 6 and Fig. 7 are shown in equations (5) and (6).

$$y = 99.071\, x^{2.0783} \quad (5)$$

$$y = 77.278\,x^{2.1448} \quad (6)$$

Here, x is the document size and y is the time taken on CPU for performing the plagiarism detection. As expected, both of them show a quadratic relationship between the size of documents and the time taken for processing the documents on CPU with the least residual error. The fitted curves are slightly deviated from execution time curves due to the variation in document size. Overall, with the increased size of documents, the processing time for plagiarism detection is increased in CPU while the time is reduced in GPU. In CPU, the number of comparisons is raised with respect to the size of the document and it executes in a sequential manner. Meanwhile, GPU handles the comparison in parallel and processing time is not affected by the increased size of documents. As a result, the execution time on GPU is reduced with the increased size of the document.

TABLE I. SPEEDUP BETWEEN CPU AND CUDA

	Execution Time (s) on GPU	Execution Time (s) on CPU	Size (MB)	No of Combinations	Speed up
Dataset-1	3563	16355	12.1	19900	5
Dataset-2	6725	43603	20.3	19900	6

Table 1 illustrates the speed up of plagiarism detection between CPU and GPU and the execution time for both datasets with 200 documents on both the platforms. Execution of plagiarism detection is around five and six times faster for dataset-1 and dataset-2 respectively on the GPU platform compared to the CPU platform. Even though, both datasets have the same number of documents, the size of dataset-2 is larger than dataset-1. Therefore, we gain better speedup when the size of the document is increased.

VI. Conclusion

Plagiarism detection is one of the time-consuming processes when it comes to handling large amount data on CPU. Moreover, the number of combinations and size of data are major, dominating factors in the execution time of the algorithm. Even though our algorithm has more string comparison than other mathematical operations, we have gained around six times speedup on GPU compared to CPU platform. Therefore, GPU platform is a more suitable environment to test a large amount of data in an enormous database for plagiarism detection. Furthermore, we can speed up the process more by using latest graphics cards.

VII. Acknowledgement

We would like to thank NVIDIA for their support in conducting this research through their GPU Research Centre Programme at the University of Peradeniya.

References

[1] Lukashenko, R., Graudina, V. and Grundspenkis, J, "Computer-based plagiarism detection methods and tools: an overview," International Conference on Computer Systems and Technologies-CompSysTech'07, ACM ISBN: 978-954-964-50-9, 2007.

[2] Butakov S, Scherbinin.V, "The toolbox for local and global plagiarism detection," Computer and Education, Volume 52, Issue 4, pp.781-788, May 2009.

[3] Lutz Prechelt, Guido Malpohl and Michael Philippsen, "Finding plagiarisms among a set of programs with JPlag," Journal of Universal Computer Science, vol. 8, no. 11 1016-1038, 2002.

[4] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," In Proceedings of SIGMOD '03, pp. 76–85, 2003.

[5] Jurriaan Hage, Peter Rademaker and Nike van Vogt, "A comparison of plagiarism detection tools," Technical Report UU-CS-2010-015, 2010.

[6] M. S. Joy and M. Luck, "Plagiarism in programming assignments," IEEE Transactions on Education, 42(2):pp.129–133, 1999.

[7] Thomas Lancaster, Fintan Culwine, "Classification of Plagiarism Detection Engines," Higher Education Academy Subject Network for Information & Computer Sciences, 2005.

[8] MAC Jiffriya, MAC Akmal Jahan and Roshan G. Ragel, "Plagiarism Detection on Electronic Text based Assignments using Vector Space Model", *7th International Conference on Information and Automation for Sustainability*, 2014.

[9] Critian Grozea, Zorana Bankovic and Pavel Laskov, "FPGA vs Multi-Cores CPU vs. GPUSs Hands on experience with a sorting Application", *Springer-Verlag Berlin, Heidelberg*, 2010.

[10] Li, Ping, Anshumali Shrivastava, Joshua L. Moore, and Arnd C. König. "Hashing algorithms for large-scale learning." In Advances in neural information processing systems, pp. 2672-2680. 2011.

[11] Sindhu L, Bindu Baby Thomas and Sumam Mary Idicula. "A Study of Plagiarism Detection Tools and Technologies", *International Journal of Advanced Research In Technology, IJART*, Vol. 1 Issue 1, pp.64-70, 2011.

[12] Xinpan YUAN, Jun LONG, Hao ZHANG, Zuping ZHANG and Weihua GUI, "Optimizing a Near-duplicate Document Detection System with SIMD Technologies", Journal of Computational Information Systems 7:11(2011) 3846-3853

[13] Lefteris Moussaiades and Athena Vakali, "Pdetect: A clustering Approach for Detecting Plagiarism in Source Code Datasets", *The Computer Journal, vol. 48 No.6*, 2005.

[14] Cheng-Hung Lin., Chen-Hsiung Liu and Shih-Chie Chang, "Accelerating Regular Expression Matching Using Hierachical Parallel Machines on GPU" *IEEE Global Telecommunications Conference (GLOBECOM),* 2011.

[15] MAC Jiffriya, MAC Akmal Jahan, Roshan G Ragel and Sampath Deegalla, "AntiPlag: plagiarism detection on electronic submissions of text based assignments", 8th International Conference of Industrial Information System (ICIIS), December 2013.

[16] David Chi Chung Tam and Mark Fiala, "A real time Augmented Reality System using GPU Acceleration", *IEEE 9th Conference on Computer and Robot Vision"*, 2012.

[17] Roberto Uribe-Paredes, Pedro Valero-Lara, Enrique Arias, Jose L Sanchez and Diego Cazorla, "Similarity Search Implementation for Multi-core and Many-core Processors", *International Conference on High Performance Computing and Simulation (HPCS),* 2011.

[18] Michael Tschuggnall and Gunther Specht, "Plagiarism detection in text document through grammar analysis of authors", *GI-Fachtagung Datenbanksysteme für Business, Technologie und Web, Magdeburg,* 2013.

[19] Hasindu Gamaarachchi, Roshan Ragel, and Darshana Jayasinghe, "Accelerating Correlation Power Analysis Using Graphics Processing Units (GPUs)", *IEEE 7th International Conference on Information and Automation for Sustainability (ICIAfS),* 2014.

[20] David B.Kirk and Wen-mei W.Hwu, "Programming Massively Parallel Processor A hand on Approach", Published by Elsevier Inc, ISBN: 978-0-12-381472-2, 2010.

[21] J. Luitjens and S. Rennich, CUDA Warps and Occupancy, *GPU Computing Webinar NVIDIA Corporation,* 2011.

[22] Benno Stien, [Online]. Available at http://www.uni-weimar.de/en/media/chairs/webis/corpora/

[23] Cheng-Hung Lin, Sheng-Yu Tsai, Chen-Hsiung Liu, Shih-Chieh Chang and Jyuo-Min Shyu, "Accelerating String Matching Using Multi-threaded Algorithm on GPU", *IEEE Globecom proceedings,* 2010.

[24] Craig Ulmerb, Maya Gokhale, Brian Gallagher, Philip Topa, Tina Eliassi-Rad, "Massively parallel acceleration of a document-similarity classifier to detect", *Journal of Parallel and Distributed Computing*, 71 (2011) 225–235.

View publication stats