Alternative CPU and GPU Parallel Computing Approaches for Improving Sequential Analysis of Probability Associations in Short Texts

Dima Alnahas and Ahmet Arif Aydin

***Abstract*—In linguistics, probabilistic relation between co-occurrent words can provide useful interpretation of knowledge conveyed in a text. Connectivity patterns of vectorized representation of lexemes can be identified by using bigram models of word sequences. Similarity assessment of these patterns is performed by applying cosine similarity and mean squared error measures on word vectors of probabilistic relation matrix of text. Moreover, parallel computing is another important aspect for various domains that enables fast data processing and analytics. In this paper, we aim to demonstrate the benefit of parallel computing for computational challenges of extracting probabilistic relations between lexemes. In this study, we have explored performance limitations of sequential semantic similarity analysis and then implemented CPU and GPU parallel versions to show benefits of multicore CPU-GPU utilization for computationally demanding applications. Our results indicate that the alternative parallel computing implementations can be used to significantly enhance performance and applicability of probabilistic relation graph models in linguistic analyses.**

***Index Terms*—Text similarity, probability relations, parallel computing, CUDA, multicore processing, GPU.**

I. Introduction

PARALLEL COMPUTING is a very important concept in various domains for a variety of tasks since it aims efficient use of underlying hardware, decreasing processing time and saving existing resources. Many research efforts have been directed towards applying multiple computer resources to compute parallel versions of sequential tasks [1], [2], [3]. Moreover, various data processing tools such as Apache Spark , Apache Pig and Apache Hadoop [4] have been developed to provide parallel computing by utilizing Google's MapReduce paradigm. Furthermore, parallel processing is highly demanded in solving computing problems that require real-time solutions since the value of gleaned information is inversely proportional [5] with processing time for real-time data analytics [6]. Thus, spending less time is crucial to perform on-demand actions for fulfilling near real-time requests [7].

Due to increasing demand for computing power in Artificial Intelligence (AI), parallel computing has been applied in vast research efforts in AI and related research areas such as Natural Language Processing (NLP), Robotics, Machine Learning, Data Mining, etc. [8]. Numerous NLP techniques facilitate information retrieval and textual patterns analyses in short texts [9] with the purpose of enhancing the potential of semantic relations extraction and its applications in Linguistics. Moreover, in natural language processing, parallel processing techniques were proven to be effective for enhancing performance of applications such as lexical analysis and shallow parsing [10]. In lexical analysis, a probabilistic graph model can be a useful tool to analyze relations among word sequences in a given text since it allows representation of these relations with low complexity. Thus, this model can be attainable by calculating words co-occurrence probabilities, which can convey semantic features and grammatical structure of text while reducing repeated lexical relations. Further exploration of language characteristics obtained from probabilistic associations of lexemes can provide more insights on relational similarity among words in short texts. In addition, vector representation of probabilistic associations among words allows for utilization of Cosine Similarity (CS) and Mean Squared Error (MSE) measures to perform relational similarity analysis [11].

One of the main purposes of this study is to apply parallel computing to perform fast and scalable relational similarity analysis on short texts of various lengths. In this paper, first, a sequential version of relational similarity analysis has been implemented. The computational complexity of relational similarity analysis and probabilistic graph model increases in proportion to the number of lexemes in the given text hence nodes in graph model and consequently imposes additional cost on time requirement of traditional serial computing. Next, one CPU and one GPU parallel versions have been implemented to get benefit of parallel computing and to decrease data processing time. These parallel versions provide a significant decrease in run time required to perform relational similarity analysis regarding the sequential version. Last, performance evaluations are presented by comparing the sequential version with the proposed CPU and GPU parallel computing approaches on the same text data.

This paper is organized as follows. In section II, a related work is presented and in section III, the methodology of the probabilistic relation graph model is explained. Then,

Dima Alnahas is with the Department of R&D, Infina Software Inc., Istanbul, TURKEY e-mail:dalnahas@infina.com.tr
orcidID: 0000-0002-6046-1066

Ahmet Arif Aydin is with the Department of Computer Engineering, Engineering Faculty, Inonu University, Malatya, 44000 TURKEY e-mail: arif.aydin@inonu.edu.tr
orcidID: 0000-0002-4124-7275

Manuscript received Feb 07, 2022; accepted Sep 30, 2022.
DOI: 10.17694/bajece.1069152

in section IV, our implementations of the sequential and two parallel version approaches are presented. In section V, computational comparisons and results of our implementations and evaluations are provided and in section VI, a conclusion is provided to present the contributions of our work.

II. Related Work

In many studies, co-occurrence probabilities of lexemes and its vector representation have been utilized in natural language processing. Also, Statistical Language Modeling is considered a successful approach for various tasks of NLP such as, machine translation, text classification, spelling correction, etc. However, similar approaches involve intensive matrix computations and analyses, thus, requiring immense computation time, power usage, and resources.

The first use of word co-occurrence probabilities in language modeling dates back to 1999. I. Dagan et al. [12] utilized a probabilistic word association model in tasks of language modeling and pseudo-word disambiguation.

In a recent work, A. Schakel and B.J. Wilson [13] introduced the use of word co-occurrence and vector representation as a significant factor of word in corpus. This study further explores the language features that can be conveyed by Word2vec [14].

In a later study, D. Alnahas and B.B. Alagoz [11] suggested a deep relational similarity analysis which explores path probabilities between words by utilizing power of probabilistic relation matrix. More recently, Y. Yin et al. [15] introduced a method to improve accuracy of text recommendation by 8.63%. The method in [15] utilizes improved cosine similarity measure to compare correlation coefficients vectors of related texts.

Moreover, in an attempt to minimize the computational cost, Mikolov et al. [16] presented the Skip-gram Model which utilizes probability to predict surrounding words in a short text. This study suggests training the Skip-gram model with distributed representations of words as a solution to achieve learned representation of phrases with minimal computational complexity.

In [17], authors accelerated text clustering speed while performing text similarity measurement by utilizing Spark architecture in parallel computing. In further effort to minimize computational cost, many researchers have explored the possible utilization of CPU and GPU cores. In a performance analysis study, S. Gupta and M.R. Babu [10] demonstrated that a 16-core GPU performs expectedly better than single-core and multi-core CPUs in the simple task of string matching. Furthermore, in a more recent study, E. Strubell et al. [18] described the financial and environmental impact of training state-of-the-art NLP models using large computational resources. This study compares the carbon emissions from training common NLP models to familiar consumptions such as, Air travel. As a result, E. Strubell et al emphasizes the need for NLP models that can be trained and developed on more affordable computational resources such as commodity laptop or server, while providing state-of-the-art analysis. As a result, these studies inspired our research to provide faster alternatives to existing NLP models and training methods by efficiently applying available hardware resources in similarity-based NLP analyses.

III. Methodology

In this section, first, a probabilistic relation graph model approach is presented for relational similarity analysis of short texts. Then, two similarity measures are applied to evaluate similarity level of probabilistic relations of word pairs. Last, an illustrative example is provided to demonstrate probabilistic associations analysis in short text.

A. Probabilistic Relation Graph Model of Short Text for Semantic Similarity Analysis

In linguistics, a word sequence of finite length can be interpreted to a form of knowledge or information. Also, word sequences can be depicted as messages and a series of messages represents a text. A vocabulary set of a message collection consists of lexical instances of message elements. Adjacent words in a message are considered to have bigram relation. The bigram relation frequency matrix of co-occurrent word pairs conveys information of co-occurrence frequency of words instances in vocabulary set. Let us form a vocabulary set of message series $M_1, M_2, .., M_h$ as,

$$W_c = w_i : w_i \perp M_1 \vee w_i \perp M_2 \vee ... \vee w_i \perp M_h \,, \tag{1}$$

where the occurrence operator $\perp$ infers that w_i item is an element of M_j message in the term $w_i \perp M_j$. The bigram relation frequency matrix of a message M is constructed by

$$R_f = \begin{cases} f_{i,j} = f_{i,j} + 1 \,, & "w_i w_j" \perp M \wedge w_i, w_j \in W_c \\ f_{i,j} = f_{i,j} & otherwise \end{cases} \tag{2}$$

Probabilistic associations between co-occurrent lexeme pairs in finite-length word sequences can be expressed by a probabilistic bigram relation graph model. The probabilistic relation matrix of bigram model is identified as normalized values of R_f elements in a range of $[0, 1]$,

$$R_p \cong \frac{1}{\sum R_f} R_f \,, \tag{3}$$

where $\sum R_f$ is summation of R_f elements and is calculated by $\sum_{i=1,j=1}^{k,k} f_{i,j}$. Each element of probabilistic relation matrix conveys the probability of relation between i^{th} and j^{th} elements of vocabulary set. Accuracy of estimating the co-occurrence probability of two lexeme items increases in proportion to length of text.

Fig. 1 illustrates a message example of word sequence $M = "w_1 w_2 w_3 w_4 w_2 w_4"$. In this figure, probabilistic transitions between co-occurrent lexeme items in word sequence M are demonstrated by probabilistic relation matrix R_p and its corresponding weighted graph representation. Zero value of probabilistic relation matrix element $p_{i,j}$ depicts the absence of co-occurrence between w_i and w_j items of text.

$$R_p = \begin{bmatrix} 0 & p_{1,2} & 0 & 0 \\ 0 & 0 & p_{2,3} & p_{2,4} \\ 0 & 0 & 0 & p_{3,4} \\ 0 & p_{4,2} & 0 & 0 \end{bmatrix}$$

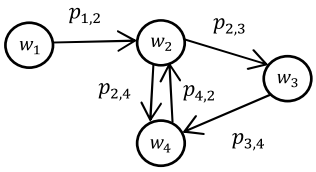


Fig. 1. A probability relation matrix of M message and corresponding weighted graph representation.

As demonstrated in Fig. 1, on one hand, the output-word vector of node w_2 is described by the 2^{nd} row elements of probabilistic matrix and is expressed as,
$u_2 = [0 \quad 0 \quad p_{2,3} \quad p_{2,4}]$.

On the other hand, the input word-vector of node w_2 is described by the 2^{nd} column elements of probabilistic matrix and is expressed as,
$v_2 = [p_{1,2} \quad 0 \quad 0 \quad p_{4,2}]^T$.

As a result, input and output word-vectors are utilized to evaluate similarity of relational transition paths between nodes. Therefore, similar probabilistic connection patterns can be an indication of relational similarity of a word pair.

B. Relational Similarity Measures Application on Vectorized Representation of Lexemes

In this study, Cosine Similarity (CS) measure is applied to assess relational similarity of lexeme pairs. CS matrix of output word-vectors can be obtained for all lexeme pairs of vocabulary set and is expressed as,

$$C_u = R_p \otimes R_p^T \,, \tag{4}$$

where the CS operator $\otimes$ performs CS calculation between vectors of R_p and R_p^T. CS matrix of input word-vectors can also be obtained for all lexeme pairs of vocabulary set and is expressed as,

$$C_v = R_p^T \otimes R_p \,. \tag{5}$$

Relational similarity based on CS is calculated with formulas 4 and 5 and can be expressed and normalized to the range of $[0, 1]$ as follows

$$C = \frac{1}{2}(C_u + C_v) \,. \tag{6}$$

Another measure to evaluate relational similarity of lexeme pairs is Mean Squared Error (MSE). Output MSE matrix is calculated for all lexemes in a vocabulary and can be expressed as,

$$E_u = R_p \ominus R_p^T \,, \tag{7}$$

where operator $\ominus$ performs MSE calculation between vectors of R_p and R_p^T. Input MSE matrix is calculated for all lexemes in vocabulary and can be expressed as,

$$E_v = R_p^T \ominus R_p \,, \tag{8}$$

MSE matrix E is then calculated by using formulas 7 and 8 for lexeme pairs and is expressed as

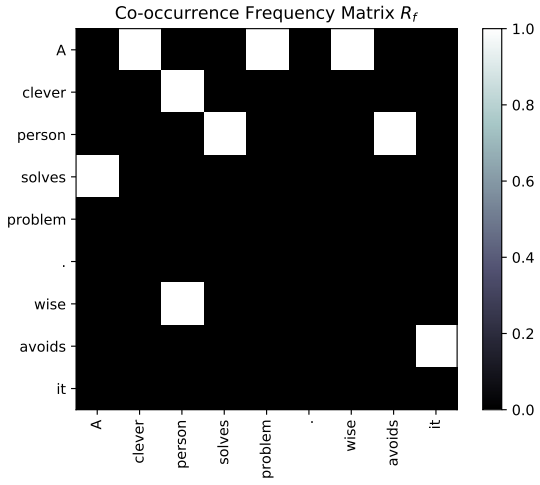
$$E = E_u + E_v \,. \tag{9}$$

C. An Explanatory Example of Probabilistic Relations Analysis in Short Text

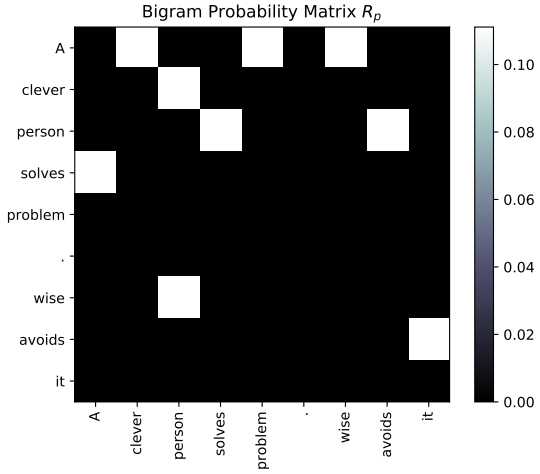
Let us consider the following text which is a quote by Einstein:
M = "A clever person solves a problem. A wise person avoids it."
This message provides the following vocabulary set:
W = A, clever, person, solves, problem, ., wise, avoids, it
Fig. 2a presents R_f matrix values of message M.
Fig. 2b shows values of R_p matrix as calculated using formula 3.
The full stop is assigned an index in the vocabulary set and it indicates the end of a sentence. As Fig. 2a demonstrates, the full stop is not included in the probabilistic calculations of associated lexemes. Fig. 3 presents bigram graph model of M. In this figure, the stream of transitions between lexemes is interrupted by full stop at the end of each sentence.
CS matrix of message M is illustrated in Fig. 4a that shows CS value of 1 for lexeme pair (wise-clever). This similarity measure indicates similar relations with similar adjacent words of the lexeme pair. In M message example, these similar adjacent words can be identified as "A" and "person" and similar relations can be detected in bigram relation graph of M.
Similarly, MSE matrix presents 0 error value for lexeme pair (wise-clever) which indicates validity of similarity analysis. The MSE matrix as calculated with formula 9 is demonstrated in Fig. 4b.
Diagonal values of CS and MSE matrices express similarity of each lexeme with itself which explains 1 values of diagonal in CS matrix and 0 values of diagonal in MSE matrix.

IV. Implementation

In this section, sequential, CPU parallel and GPU parallel versions are explained.



(a) R_f values of M message.



(b) R_p values of M message.

Fig. 2. R_f and R_p values of M message.

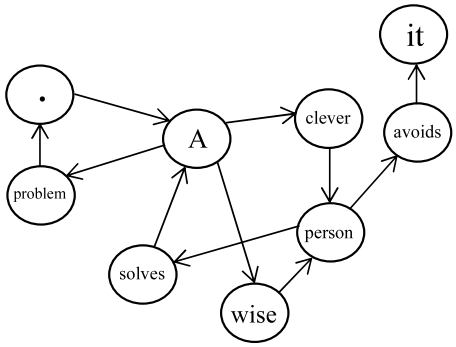
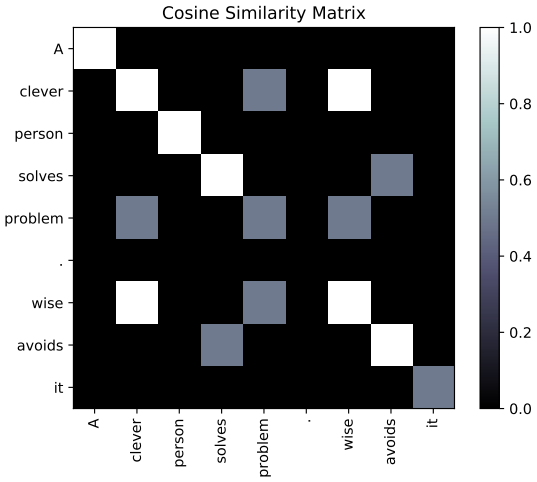
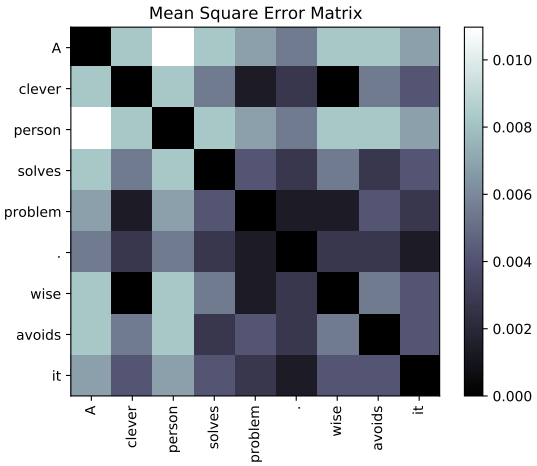


Fig. 3. Bigram relation graph of M message.



(a) CS matrix of M message.



(b) MSE matrix of M message.

Fig. 4. CS and MSE matrices of M message.

A. *Sequential Probabilistic Similarity Analysis*

Pseudocode of the sequential version of probabilistic similarity analysis algorithm is provided in Algorithm IV-A.

Pre-process text file
Create dictionary W of text
for each element i of W **do**
  Obtain l list of adjacent elements to i
  for each element j of l **do**
    $R_f(i,j) \leftarrow R_f(i,j) + 1$
Total frequency $T \leftarrow$ sum of R_f elements
for each element p of R_p and corresponding element f of R_f **do**
  $p \leftarrow \frac{f}{T}$
Calculate Euclidean norm vector N_u of R_p
Norms multiplication matrix $N_{u_{matrix}} \leftarrow N_u \times N_u^T$
for each n_u element of $N_{u_{matrix}}$ **do**
  if $n_u = 0$ **then**
    $n_u \leftarrow 1$
for each element u of C_u and corresponding elements n_u of $N_{u_{matrix}}$, p of R_p and p_T of R_p^T **do**

$u \leftarrow \frac{(p \times p_T)}{n_u}$
Calculate Euclidean norm vector N_v of R_p^T
Norms multiplication matrix $N_{v_{matrix}} \leftarrow N_v \times N_v^T$
for each n_v element of $N_{v_{matrix}}$ **do**
  if $n_v = 0$ **then**
    $n_v \leftarrow 1$
for each element v of C_v and corresponding elements n_v of $N_{v_{matrix}}$, p of R_p and p_T of R_p^T **do**
  $v \leftarrow \frac{(p \times p_T)}{n_v}$
for each element c of C and corresponding elements u of C_u and v of C_v **do**
  $c \leftarrow \frac{(u+v)}{2}$
Calculate R_p size S
for each $e_u(x_u, y_u)$ element of E_u and corresponding x_u and y_u vectors of R_p **do**
  $e_u(x_u, y_u) \leftarrow$ sum of $(x_u - y_u)^2$ elements $/S$
for each $e_v(x_v, y_v)$ element of E_v and corresponding x_v and y_v vectors of R_p **do**
  $e_v(x_v, y_v) \leftarrow$ sum of $(x_v - y_v)^2$ elements $/S$
for each element e of E and corresponding elements e_u of E_u and e_v of E_v **do**
  $e \leftarrow e_u + e_v$

Fig. 5 illustrates serially computed matrices in the sequential version of the algorithm as explained in section III-C.

To obtain co-occurrence matrix R_f for probabilistic similarity analysis, words associations of M-length text are explored sequentially to construct a dictionary of lexeme items in text. For a dictionary of N elements, indexes of dictionary lexemes are used as columns and rows index of R_f. Therefore, R_f matrix conveys frequency information of $N \times N$ possible lexeme pair co-occurrences. Inspecting relational connectivity of word pairs with window size of 2 in M-length text requires $M - 1$ iterations in order to construct R_f matrix. To reduce computing time of frequently adjacent lexeme pairs in textual patterns such as grammatical structures, this study implies to alternatively iterate dictionary elements and obtain adjacent words list with corresponding occurrence frequency for each item in the dictionary. The inferred computing sequence can significantly reduce execution time particularly for long texts that produce relatively small vocabulary sets.
Probabilistic relation matrix R_p is populated by using R_f as expressed in formula 3. R_p describes association probability between each two items in dictionary, thus, it is obtainable by calculating co-occurrence probability of $N \times N$ word pairs.
CS matrices of input and output word-vectors of R_p and R_p^T are computed discretely to obtain CS matrix as expressed in formula 6. Each element of both CS matrices is a depiction of cosine the angle between input and output vectors of word pairs. Similarly, MSE matrices of input and output word-vectors of R_p and R_p^T requires sequential computing of Euclidean distance between input and output word-vectors of $N \times N$ word pairs.

B. Parallel Computing Approaches for Probabilistic Similarity Analysis

In this section, one CPU and one GPU parallel approaches are presented. These approaches have been developed to efficiently explore information extraction properties of connectivity analysis for long texts in parallel by making use of parallel computing tools. CPU and GPU parallel approaches are respectively explained next.

1) Multicore CPU Version: In this parallel CPU version, the Multiprocessing module of Python programming language is utilized to parallelize the process of extracting connectivity features of word sequences. The Multiprocessing module contains the Pool class which automatically initiates processes as many as core number of CPU when process number is not deliberately specified. Using the Map function, this class can divide elements of the argument matrix between the spawned processes which simultaneously execute the specified task by making use of underlying CPU cores. The computing steps of the CPU parallel version is provided in Fig. 6 that also presents how matrices are computed in this parallel version of probabilistic similarity analysis algorithm.

In this parallel CPU version, two main portions of the sequential algorithm have been parallelized. First, obtaining CS matrix for input and output word-vectors of R_p and R_p^T implemented by making use of Multiprocessing Pool class that drastically reduces time complexity of this task. Alternative to iterating matrix elements sequentially, elements are divided into chunks between initialized processes. Then, processes concurrently execute serial instructions of computing CS of lexeme pairs to construct CS matrix. The second main parallelized portion of the sequential algorithm is computing MSE matrix for input and output word-vectors of R_p and R_p^T. In this part word-vector pair chunks are distributed among processes which discretely calculate the MSE index of each pair.

2) A GPU Version: In this section, a GPU version of probabilistic similarity analysis is presented. Modern graphic processors have introduced drastic solutions for immensely large computation problems. Also, many research efforts continue to investigate the prospect of utilizing GPUs to reduce time complexity of analysis tasks in fields of deep learning and information processing [19]. In particular, CUDA (Compute Unified Device Architecture) programming model has been utilized to establish the required integration of NVIDIA's GPU with multicore CPU for parallel processing purposes. This parallel computing platform applies Single Instruction Multiple Thread (SIMT) execution model to manage and schedule warps independently, hence concurrently [20]. Moreover, instruction of threads can be specified by a C function that denotes a kernel that is executed concurrently by all available threads in the instruction sequence. Threads are organized in grids which consist of thread blocks. Each block is assigned a shared memory space that can be accessible by 512 threads within the block [21] and up to 1024 threads with recent CUDA toolkit.
In this paper, we aimed to explore CUDA's potential on concurrent extraction of textual features in near real-time manner. With the aid of CUDA, time complexity of obtaining similarity measures and probabilistic connectivity matrices of text is significantly reduced by efficiently expressing word

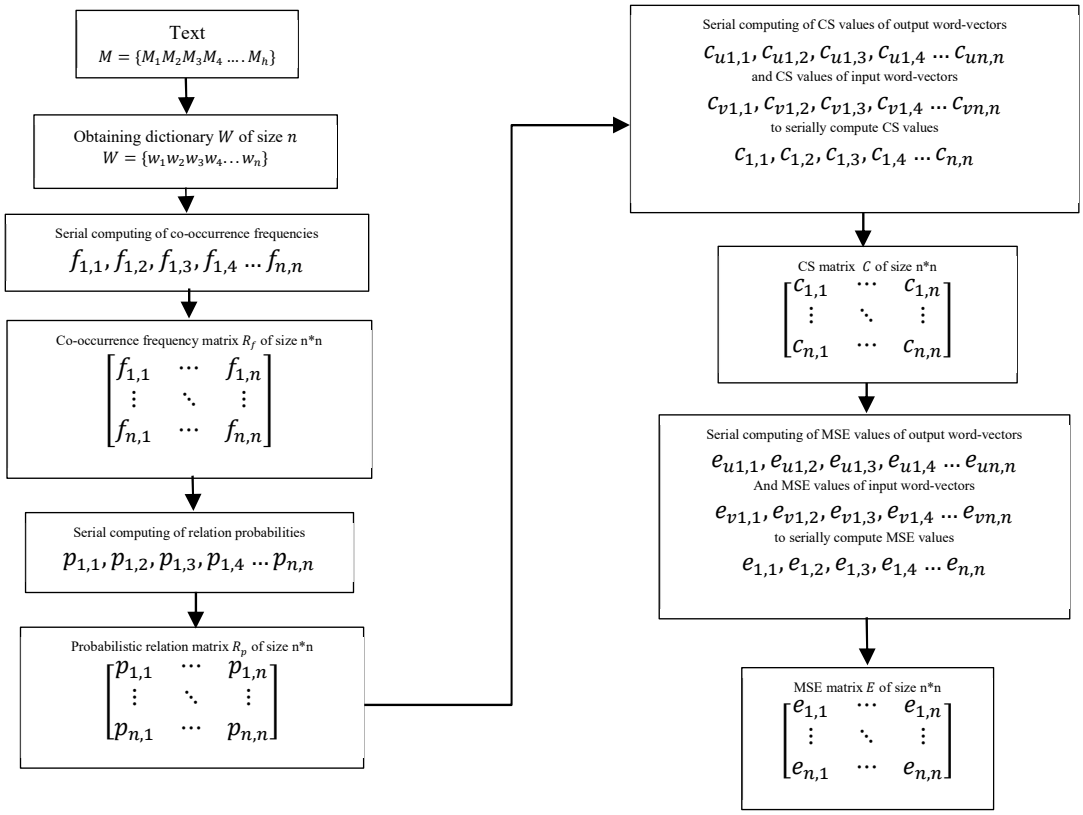


Fig. 5. Computing steps of the serial version of similarity analysis.

sequences in blocks. At least eight blocks per multiprocessor can be executed simultaneously in association with hardware limitations and memory resources.

Fig. 7 presents possible blocks division of processed matrices in GPU version of similarity analysis. Each element of these blocks corresponds to a thread which computes one element of the processed matrix. Block number is usually assigned in accordance to data size.

Pseudo code of the GPU version of probabilistic similarity analysis algorithm is presented in Algorithm IV-B2.

Pre-process text file
Create dictionary W of text
for each element i of W **do**
  Obtain l list of adjacent elements to i
  for each element j of l **do**
    $R_f(i,j) \leftarrow R_f(i,j) + 1$
Total frequency $T \leftarrow$ sum of R_f elements
Initialize block dimensions $B_{dim}(x_{thread}, y_{thread}, z_{thread})$, total thread number $t \leftarrow x_{thread} + y_{thread} + z_{thread}$
Initialize grid dimensions $G_{dim}(x_{block}, y_{block}, z_{block})$, total block number $t \leftarrow x_{block} + y_{block} + z_{block}$
Divide R_p matrix to b blocks
for each block **in parallel do**
  for each element p of R_p and corresponding element f of R_f **do**
    $p \leftarrow \frac{f}{T}$
Calculate Euclidean norm vector N_u of R_p
Norms multiplication matrix $N_{u_{matrix}} \leftarrow N_u \times N_u^T$
Divide $N_{u_{matrix}}$ and C_u matrices to b blocks
for each block **in parallel do**
  for each n_u element of $N_{u_{matrix}}$ block **do**
    if $n_u = 0$ **then**
      $n_u \leftarrow 1$
  for each element u of C_u and corresponding elements n_u of $N_{u_{matrix}}$ block , p of R_p block and p_T of R_p^T block **do**
    $u \leftarrow \frac{(p \times p_T)}{n_u}$
Calculate Euclidean norm vector N_v of R_p^T
Norms multiplication matrix $N_{v_{matrix}} \leftarrow N_v \times N_v^T$
Divide $N_{v_{matrix}}$ and C_v matrices to b blocks
for each block **in parallel do**
  for each n_v element of $N_{v_{matrix}}$ block **do**
    if $n_v = 0$ **then**
      $n_v \leftarrow 1$
  for each element v of C_v and corresponding elements n_v of $N_{v_{matrix}}$ block , p of R_p block and p_T of R_p^T block **do**
    $v \leftarrow \frac{(p \times p_T)}{n_v}$
for each element c of C and corresponding elements u of C_u and v of C_v **do**
  $c \leftarrow \frac{(u+v)}{2}$
Calculate R_p size S
Divide E_u to b blocks
for each block **in parallel do**
  for each $e_u(x_u, y_u)$ element of E_u block and corresponding x_u and y_u vectors of R_p block **do**
    $e_u(x_u, y_u) \leftarrow$ sum of $(x_u - y_u)^2$ elements$/S$
Divide E_v to b blocks
for each block **in parallel do**
  for each $e_v(x_v, y_v)$ element of E_v block and corresponding x_v and y_v vectors of R_p block **do**
    $e_v(x_v, y_v) \leftarrow$ sum of $(x_v - y_v)^2$ elements$/S$
for each element e of E and corresponding elements e_u of E_u and e_v of E_v **do**
  $e \leftarrow e_u + e_v$

V. Results and Evaluations

In this section, performance evaluations of sequential and parallel versions of probabilistic similarity analysis approaches are presented. In addition, Fig. 8 demonstrates performance

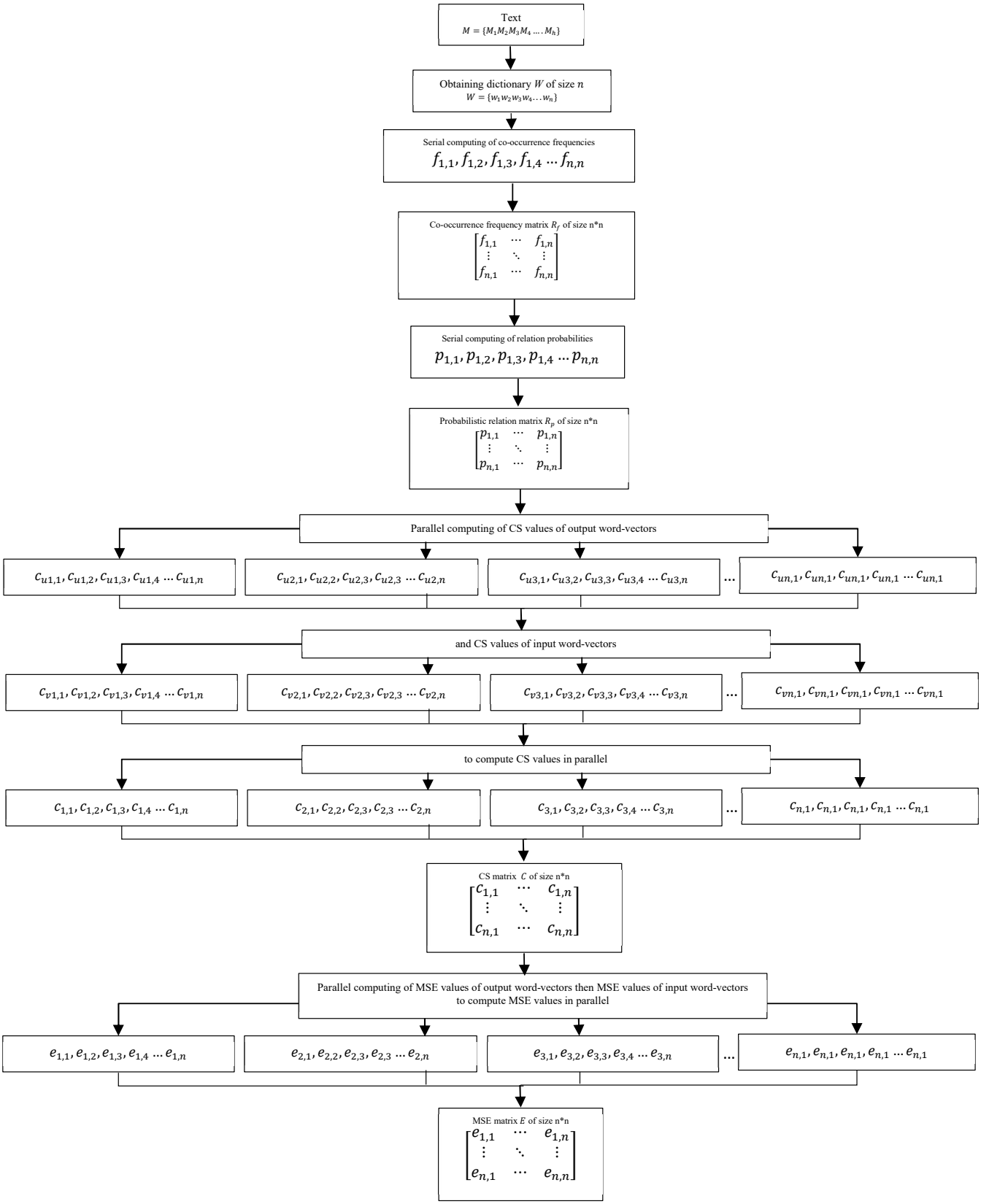


Fig. 6. Computing steps of CPU parallel version of similarity analysis.

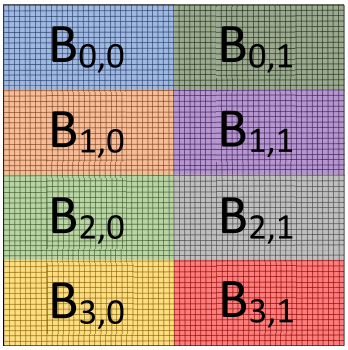


Fig. 7. Possible 8-block $B(i, j)$ division of computed matrices in GPU version of similarity analysis.

assessments of the sequential version, the CPU parallel version that utilizes Python's Multiprocessing module and Pool class, and the GPU version using CUDA.

These algorithms are tested on a PC with 16GB RAM; in the GPU parallel version, NVIDIA GeForce GTX 960M (4 GB, 640 cores, GDDR5, 1253 MHz) laptop graphic card was utilized. Additionally, the time spent for device-to-host and host-to-device data transfer was also included in the calculated time costs. Moreover, in the CPU parallel version, Intel Core i7-6700HQ CPU (2.60GHz) was used and each performance test is run with 8 threads (1 thread per core).

In this performance evaluation we used the Blog Authorship corpus, which consists of posts gathered from 19,320 bloggers on blogger.com. The corpus incorporates a total of 681,288 posts and over 140 million words [22].

Due to hardware resource limitations and the nature of the probabilistic similarity analysis algorithm, the serial, CPU parallel and GPU parallel versions of algorithms have reached an upper limit for processed text length that are respectively 73000 words for serial version, 134000 words for the CPU parallel, and 313000 words for the GPU parallel version.

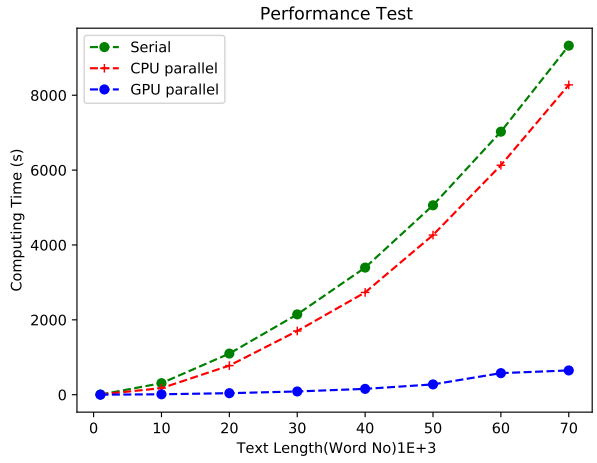


Fig. 8. Computing time of GPU, CPU parallel and serial execution of probabilistic similarity analysis for various text lengths.

Fig. 8 demonstrates the performance comparison of serial, CPU parallel and GPU parallel algorithms by conducting probabilistic similarity analysis. Each algorithm performed analyses on the same text and number of words increased during each iteration by 10K. For this case, the maximum text length is 70K words due to the serial version's upper limit.

The sequential version that serially attains necessary measures and extracts connectivity features of text scores lower computing time than the parallel CPU version for text length less than 2K words. In serial version, computing time then starts increasing drastically for larger data, illustrating quadratic complexity of similarity analysis in accordance with vocabulary set, hence length of text.

The CPU parallel version performs poorly with regard to computing time for text with word number less than 2K since forking data across CPU cores and then joining results of parallel threads creates additional time cost and overheads. This result confirms that using parallel processing and multicore CPUs is not always guaranteed to provide speedup for all sizes of datasets. In order to get benefit of parallel processing, the amount of processed data needs to be increased to monitor speedup and gain performance as shown in Fig. 8.

The GPU version of the analysis shows best performance in Fig. 8 for all text lengths. This outcome is produced by instant computing of multiple data blocks simultaneously without lost computing time for initialization procedures as seen in CPU parallel version. Utilizing GPU cores by CUDA for conducting probabilistic similarity analysis offers minimum processing time in comparison to other versions of the algorithm regardless of data size, hence reducing proportional association between text length and computing time.

Table I shows computing time values that are utilized to create the Fig. 8.

TABLE I
COMPUTING TIME VALUES OF SERIAL, CPU PARALLEL AND GPU PARALLEL VERSIONS OF ANALYSIS.

Word Count	Sequential	CPU Parallel	GPU Parallel
1K-word	4.67	8.17	0.5
2K-word	17.66	14.45	0.93
10K-word	307.95	174.79	11.52
20K-word	1098.58	777.61	40.56
30K-word	2146.1	1701.46	86.62
40K-word	3395.67	2732.19	155.94
50K-word	5060.01	4265.16	275.19
60K-word	7026.09	6129.2	576.18
70K-word	9324.22	8276.45	648.48

To conclude, Table I shows that the serial version is 1.7 times faster than the CPU parallel version and the GPU parallel CUDA version is 9.3 times faster than the serial version for text length of 1K words. On the other hand, the CPU parallel version is at least 1.2 times (2K words) and at most 1.1 times (70K words) faster than the serial version. The GPU CUDA version is 14.4 times faster than the sequential version. Also, the GPU version is 12.8 times faster than the CPU parallel version for text length of 70K words.

To further explore the results of the CPU parallel and GPU parallel versions, time cost of both versions for text length bigger than 70K words have been processed as shown in Fig. 9. These results are obtained by conducting probabilistic similarity analysis for CPU and GPU parallel versions of the algorithm on the same text starting from 80K to 130K-

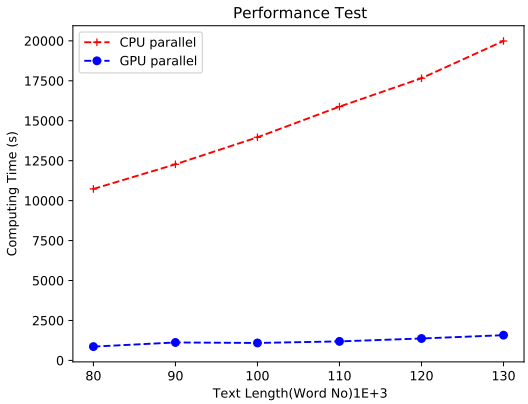


Fig. 9. Computing time of GPU and CPU parallel execution of probabilistic similarity analysis.

TABLE II
COMPUTING TIME VALUES OF SERIAL, CPU PARALLEL AND GPU PARALLEL VERSIONS OF ANALYSIS.

Word Count	CPU Parallel	GPU Parallel
80K-word	10731.88	863.31
90K-word	12273.78	1122.23
100K-word	13963.98	1091.58
110K-word	15881.9	1189.49
120K-word	17660.96	1370.83
130K-word	19990.45	1578.51

word length. Similar to analysis in Fig. 8, each algorithm processed the same data and the number of processed words was increased by 10K words during each iteration.

Table II demonstrates that the GPU parallel version is at least 12.43 times (80K words) and at most 12.66 times (130K words) faster than the CPU parallel version.To sum up, these results show that parallel versions of probabilistic similarity analysis algorithms are promising to utilize applications of similarity based NLP text analysis.

VI. CONCLUSION

In this study, we presented a semantic similarity analysis to indicate relevance of co-related words in text. In linguistics, frequency of co-related lexemes in short text can be useful to utilize probabilistic features of connectivity patterns for semantic elicitation of knowledge conveyed in text. Probabilistic associations of word pairs provide an insight to the textual structure of lexeme sequences. CS and MSE measures can be obtained from input and output-word vectors to denote probabilistic relational similarity of word vector pairs. Moreover, parallel computing is another important aspect in data processing and analytics since the concept of parallel computing has been applied in semantic similarity analysis. In this paper, first, a sequential version is proposed, and then a CPU parallel version is developed and last a GPU parallel CUDA version implemented to get benefits of parallel processing for probabilistic semantics analysis. The results presented in section V indicate that performance limitations of serial similarity analysis are significantly reduced by proposed CPU parallel and GPU parallel versions. Furthermore, this study infers efficiency of utilizing parallel processing techniques and applying graphic processor resources to expand capacity of analyzing probabilistic relations and its indication of similarity analysis among lexemes.

REFERENCES

[1] A. A. Aydin and G. Alaghband, "Sequential and parallel hybrid approach for nonrecursive most significant digit radix sort," in *10th International Conference on Applied Computing*, 2013, pp. 51–58.
[2] S. Berkovich and E. Berkovich, "Methods and apparatus for concurrent execution of serial computing instructions using combinatorial architecture for program partitioning," Apr. 8 1997, uS Patent 5,619,680.
[3] A. A. Aydin, "Performance benchmarking of sequential, parallel and hybrid radix sort algorithms and analyzing impact of sub vectors, created on each level,on hybrid msd radix sort's runtime," 2012, mS Thesis, University of Colorado Denver.
[4] B. Parhami, "Parallel processing with big data." 2019.
[5] D. Demirol, R. Das, and D. Hanbay, "Büyük veri üzerine perspektif bir bakış," in *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*. IEEE, 2019, pp. 1–9.
[6] J. Hromkovič, *Communication complexity and parallel computing*. Springer Science & Business Media, 2013.
[7] A. Aydin and K. Anderson, "Batch to real-time: Incremental data collection & analytics platform," 2017.
[8] S. H. Roosta, "Artificial intelligence and parallel processing," in *Parallel Processing and Parallel Algorithms*. Springer, 2000, pp. 501–534.
[9] T. Strzalkowski, F. Lin, J. Wang, and J. Perez-Carballo, "Evaluating natural language processing techniques in information retrieval," in *Natural language information retrieval*. Springer, 1999, pp. 113–145.
[10] S. Gupta and M. R. Babu, "Performance analysis of gpu compared to single-core and multi-core cpu for natural language applications," *IJACSA Editorial*, 2011.
[11] D. Alnahas and B. B. Alagoz, "Probabilistic relational connectivity analysis of bigram models," in *2019 International Artificial Intelligence and Data Processing Symposium (IDAP)*. IEEE, 2019, pp. 1–6.
[12] I. Dagan, L. Lee, and F. C. Pereira, "Similarity-based models of word cooccurrence probabilities," *Machine learning*, vol. 34, no. 1, pp. 43–69, 1999.
[13] A. M. Schakel and B. J. Wilson, "Measuring word significance using distributed representations of words," *arXiv preprint arXiv:1508.02297*, 2015.
[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
[15] Y. Yin, D. Feng, Z. Shi, and L. Ouyang, "Text recommendation based on time series and multi-label information," 2020.
[16] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*, 2013.
[17] S. Zhou, X. Xu, Y. Liu, R. Chang, and Y. Xiao, "Text similarity measurement of semantic cognition based on word vector distance decentralization with clustering analysis," *IEEE Access*, vol. 7, pp. 107 247–107 258, 2019.
[18] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019.
[19] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 873–880.
[20] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *IEEE micro*, vol. 28, no. 2, pp. 39–55, 2008.
[21] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for?" *Queue*, vol. 6, no. 2, pp. 40–53, 2008.
[22] J. Schler, M. Koppel, S. Argamon, and J. Pennebaker, "Effects of age and gender on blogging. aaai spring symposium on computational approaches for analyzing weblogs," 2006.

Dr. Ahmet Arif Aydin is an assistant professor of the Computer Engineering Department at Inonu University. He earned his Ph.D. in Computer Science at the University of Colorado Boulder in 2016 with a specialization in "Architectural Design for Data Analytics Platforms". His current research interests include software engineering, data-intensive system design, crisis informatics, big data analytics, data modeling, algorithm design for analytics, parallel processing, and machine learning.

Dima Alnahas currently works as a Software Development and AI Team Leader at Infina Software Inc. She is pursuing her Masters' Degree at Computer Engineering Department in Kadir Has University. She also attends Baden-Wuerttemberg Cooperative State University (DHBW) as an exchange masters' student with research interests in Natural Language Processing and Machine Learning.