Multithreaded Sliding Window Approach to Improve Exact Pattern Matching Algorithms

Ala'a Al-shdaifat
Computer Information System Department
The University of Jordan
Amman, Jordan

Mohammad Abushariah
Computer Information System Department
The University of Jordan
Amman, Jordan

Bassam Hammo
Computer Information System Department
The University of Jordan
Amman, Jordan

Esra'a Alshdaifat
Computer Information System Department
The Hashemite University
Zarqa,Jordan

***Abstract*—In this paper an efficient pattern matching approach, based on a multithreading sliding window technique, is proposed to improve the efficiency of the common sequential exact pattern matching algorithms including: (i) Brute Force, (ii) Knuth-Morris-Pratt and (iii) Boyer-Moore. The idea is to divide the text under-search into blocks, each block is allocated one or two threads running concurrently. Reported experimental results indicated that the proposed approach improves the performance of the well-known pattern matching algorithms, in terms of search time, especially when the searched patterns are located at the middle or at the end of the text.**

***Keywords*—pattern matching; multithreading; sliding window; Brute Force; Knuth-Morris-Pratt; Boyer-Moore**

I. Introduction

In the world of Internet and the increasing availability of huge data, text remains the main form to exchange knowledge. Searching text for a "string matching", also referred to as "pattern matching" , is an active research area with respect to text processing domain. Pattern matching algorithms are the basic components used in the implementation of practical software applications existing under most operating systems. This area of research is expected to grow widely due to the increasing demand of speed associated with many applications related to pattern matching [1]. Examples of pattern matching applications include: text editors, database queries, computational molecular biology, network intrusion detection system, information retrieval, natural language processing, web search engines, language syntax checker, digital libraries, two dimensional mesh, ms word spell checker and many other applications [2]. The quantity of available data in these fields increases enormously. This is the reason why algorithms should be efficient even if the speed and capacity of storage of computers increase regularly. Pattern matching consists of finding one, or more generally, all the occurrences of a pattern in a text. More formally, the input to the pattern matching problem are: (i) the pattern P and (ii) the text T. The pattern is denoted by P = T [0 ... m-1], where m is the length of the pattern. The text is denoted by T = T [0 ... n-1], where n is the length of the text. Both strings are built over a finite set of characters called an alphabet and denoted by $\sum$ [3]. Many sequential algorithms exist for pattern matching and are widely used in practice. The most well-known ones are :

1) Brute-force exact pattern matching algorithm.
2) Knuth-Morris-Pratt (KMP) exact pattern matching algorithm.
3) Boyer-Moore exact pattern matching algorithm.

The sliding window mechanism [4] is always utilised to implement the previous sequential pattern matching algorithms. Where the text is scaned with the help of a window whose size is generally equal to m (the same as pattern size). The search process starts with aligning the left ends of the window and the text and then compare the characters of the window with the characters of the pattern p. After a complete match of the pattern or after a mismatch is reported the window will be shifted to the right. The process is repeated again until the right end of the window goes beyond the right end of the text. The main issue associated with sequential exact pattern matching algorithms is the efficiency. More specifically, as the text size increases the efficiency tends to degrade. The work presented in this paper utilises multithreading programming technique to execute the pattern matching process simultaneously in a timesharing manner. The text is divided into text blocks and assigned to threads. More specifically, each block is assigned to: (i) forward thread and (ii) backward thread. The forward thread runs in a forward direction starting from the top of the block "top-down". While backward thread starts the search process from the bottom of the block "bottom-up". Each thread scans and searches the text at different places as the string may occur anywhere within the text. The conjecture advantage is to improve the performance of the search process.

The rest of this paper is organised as follows. Section II gives a review of related work on pattern matching algorithms along with some background on sliding window technique and multithreading motivation. Section III describes the proposed multithreaded sliding window technique for pattern matching algorithms. Section IV presents an evaluation of the proposed approach as applied to a range of different data collections. Section V summarises the work and indicates some future research directions.

II. Literature Review

This section provides a review of sequential pattern matching algorithms, sliding window mechanism and multithreading. The section is organised as follows: Section II-A presents Brute-Force exact pattern matching algorithm, Section II-B provides an overview of Knuth-Morris-Pratt exact pattern matching algorithm, While II-C presents Boyer-Moore exact pattern matching algorithm. An overview of sliding window mechanism is presented in Section II-D. Section II-E provides an overview of the related work on multithreading programming technique as a solution to the efficiency problem associated with sequential pattern matching algorithms.

A. Brute-Force Exact Pattern Matching Algorithm

This section presents an overview of Brute-Force (BF) exact pattern matching algorithm. The BF algorithm is a naive algorithm that compares each character in the pattern with its corresponding character in the input text. In a case of a complete match or a mismatch of the pattern it shifts one position to the right [1]. It is worth to note that the BF algorithm has no preprocessing phase; it has only a searching phase. During the searching phase, each position in the text T is checked to see if the pattern P starts in that position. With respect to the time complexity of the searching phase, in the worst case, the time complexity is O(mn) where m is the pattern length and n is the text length [5].

B. Knuth-Morris-Pratt (KMP) Exact Pattern Matching Algorithm

In this section an overview of Knuth-Morris-Pratt (KMP) is provided. KMP algorithm was proposed by Knuth, Morris and Pratt in 1977 [6] to improve and speed up the algorithm that proposed earlier by Morris and Pratt [7]. The KMP algorithm performs a character comparison from left to right of the pattern and it avoids comparisons with the input text that has previously been involved in comparison with some element of the input pattern. This is done by using information of the previous character that has already been tested in order to decide the next sliding position. Unlike the BF algorithm, the KMP algorithm has preprocessing phase, in addition to the searching phase. Within the preprocessing phase the pattern is preprocessed to find matches of prefixes of the pattern with the pattern itself. The information obtained from the preprocessing phase is used to avoid naive BF useless shifts of the pattern, so backtracking on the string never occurs. More specifically, KMP algorithm consists of the following two functions:

1) Prefix Function. This function is used to compute the number of shifts that the pattern can be moved to avoid wasteful comparisons.
2) KMP Matcher This function takes the text, the pattern and the prefix function as inputs. The target is to find the occurrence of the searched pattern within the text.

The time complexity of the preprocessing and searching phases are Θ(m) and Θ(n) respectively, where m is pattern length and n is the text length [3].

C. Boyer-Moore Pattern Matching Algorithm

This section presents Boyer-Moore pattern matching algorithm. Boyer-Moore algorithm searches from left to right and performs character comparisons within its sliding window from right to left. The BM algorithm performs preprocessing on the pattern by using two heuristics: (i) bad-character shift and (ii) good-suffix shift. In bad-character heuristic, the input pattern is shifted to align the mismatched character with the rightmost position, where the mismatched character is placed in the input pattern. In the good-suffix, the mismatch occurs in the middle of the search string. Therefore the input pattern is shifted to the next occurrence of the suffix in the string [8]. The time and space complexity of the preprocessing phase is O(m+$|\sum|$). While the running time of the searching phase is O(nm + $|\sum|$) in the worst case. Note that m is the pattern length and n is the text length [1].

D. Sliding Window Mechanism

In this section an overview of sliding window mechanism is presented. Most pattern matching algorithms scan the text with the help of a window, whose size is generally equal to the pattern size. This mechanism is referred to as "sliding window" mechanism [4]. The general procedure is as follows. At the beginning of the search, the left end of the window is aligned with the left end of the text. Then the occurrence of the pattern is checked, this process is referred to as an "attempt". The check is generally carried out by comparing the characters of the window with the characters of the pattern [9]. After finding a match of the pattern, or after a mismatch is detected, the window is shifted to the right by a finite number of positions according to the shift strategy. The same process is repeated again until the right end of the window goes beyond the right end of the text. Recently some researchers suggested the use of multiple windows to scan the text simultaneously, in order to improve the efficiency of the search process [10], [11], [12].

E. Threads and Multithreading

This section presents an overview of: (i) threads and multithreading and (ii) utilising multithreading techniques to reduce the computation time of pattern matching approaches. With respect to threads and multithreading, the thread is the basic unit of execution [13]. In single threaded applications, all operations are executed sequentially by a single thread. More specifically, an operation must complete before the other operations can begin, also there is only one thread running at a time [14]. While in multithreading applications, multiple threads run simultaneously in a timesharing manner [15]. This allows many parts of the same program to run concurrently on a single processor system [14]. Java makes concurrency available to application programs running on a Java machine. Java programs can have multiple threads of execution, where each thread is responsible for a portion of the program that may execute concurrently with other threads while sharing with them application resources such as memory [14]. Every Java thread has a priority which helps the operating system to determine the order in which threads are scheduled. A thread with a higher priority is allocated a processor time before a lower priority one [14]. Recently, multithreading techniques have been utilised to decrease the computation time of pattern matching approaches [16], [17]. Kofahi and

Abusalama [16] suggested a framework that uses data distribution and multithreading techniques for string matching. The main idea is based on applying a multithreading technique which concurrently searches the text at different positions. The framework combines two techniques of concurrency: (i) a multithread technique that searches the target text simultaneously in a time sharing manner, in which each thread starts searching the target text at different positions, and (ii) a technique that distributes the search load among multiple servers and implements the multithreading approach on small sub text. A novel multithreaded string matching approach was proposed by Nirmala and Rajagopalan for exact occurrences of string in DNA sequences [17]. In this approach, the DNA sequence is divided into parts depending on string size, and then multiple search threads search the string simultaneously in a timesharing manner. The proposed technique depends on the pre-processing phase, which retrieves the index. Rasool et al. [18] also utilised multithreading techniques to improve the CPU utilization and increase the time efficiency for a hybrid string matching algorithm that combines Knuth-Morris-Pratt (KMP) and Boyer-Moore. The work presented in this paper utilises multithreading techniques to execute the pattern matching process simultaneously in a timesharing manner. The main idea is to split the text into text blocks with associated threads running on the blocks. Each thread scans and searches the text in different places as the string may occur anywhere within the text.

III. The Proposed Efficient Pattern Matching Approach

In this section the suggested Efficient Pattern Matching (EPM) approach is described. As noted in the introduction to this paper the proposed approach adopts: (i) sliding window mechanism and (ii) multithreading techniques. The intuition behind using multithreading techniques is that it could improve the efficiency of the common sequential exact pattern matching algorithms. The EPM approach is applied to three well-known pattern matching algorithms namely: (i) BF, (ii) KMP and (iii) BM. The rest of this section is organized as follows: Section III-A presents the proposed Efficient Pattern Matching Approach, while Section III-B provides a complete working example explaining how it works.

A. Efficient Pattern Matching (EPM) Approach

In this section the Efficient Pattern Matching (EPM) approach is described. As noted above the EPM utilises multithreading techniques to improve the efficiency of the pattern matching algorithms. The main idea is to divide the text document into blocks and assign the blocks to threads to run concurrently. Figure 1 illustrates the process. The text is divided into w blocks, and each block is allocated a "forward" thread that runs in a forward direction starting from the top of the block. In Figure 1 each block is assigned to one forward thread. In the work presented in this paper assigning a block to two independent threads is also considered. More specifically, each block is assigned to: (i) forward thread and (ii) backward thread. Figure 2 illustrates assigning each block to a forward and backward thread. Recall that forward thread runs in a forward direction starting from the top of the block "top-down". While backward thread starts the search process from the bottom of the block "bottom-up".

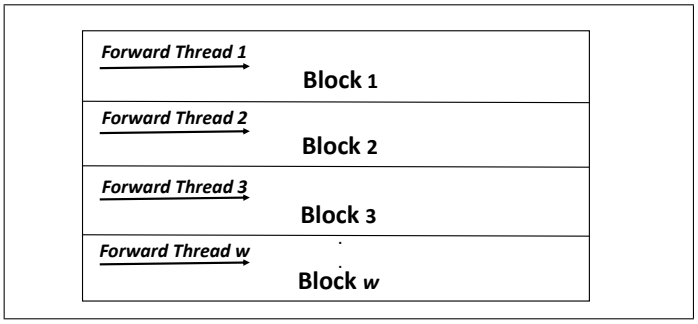

Fig. 1: A text divided into w blocks,each block is assigned to onel forward thread

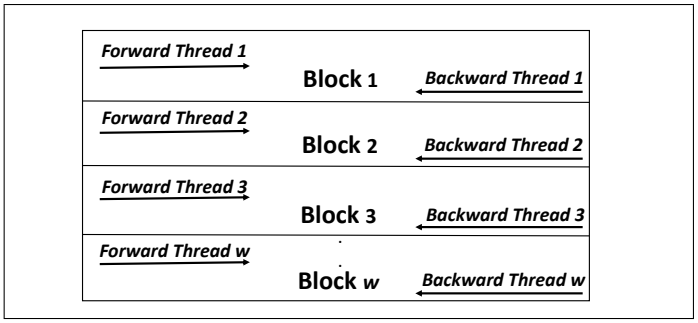

Fig. 2: A text divided into w blocks, each block is assigned to two threads: (i) a forward thread and (ii) a backward thread

It is interesting to note here that the size of each block specified according to the total number of words in the text as follows:

1) If the number of words in the text is even:
   Block size = n/w.
   Where n is the number of words in the text and w is the number of blocks.
2) If the number of words in the text is odd:
   Block size= n/w.
   The last block size = block size + the reminder of the division.
   Again, n is the number of words in the text and w is the number of blocks.

Algorithm Efficient Pattern Matching approach presents the proposed EPM procedure. The input to the algorithm is the text under search $text$ and the number of desired blocks $NumOfBlocks$. The process commences by dividing the text into blocks according to $NumOf\ Blocks$ and number of words in $text$ as explained earlier (line 10). Then calculate and allocate the required number of threads, note here that two threads (forward and backward) will be allocated for each block (line 11). After that we loop through the blocks (line 12) and on each iteration: (i) assign block i to a forward thread FT (line 13), (ii) assign block i to a backward thread BT (line 14), (iii) start search process at block i, using the specified pattern matching algorithm, with respect to FT (line 15), and (iv) start search process at block i, using the specified pattern matching algorithm, with respect to BT (line 16). The output of the EPM procedure will be the occurrences of the searched pattern in the text associated with the time consumed to find them using the forward and backward threads.

Efficient Pattern Matching approach

INPUT
$text$ The text under search
$NumOfBlocks$ The number of desired blocks

OUTPUT
The occurrences of the searched pattern in the text associated with the time consumed to find them using the forward and backward threads

$block$ A segment of text
FT Thread starts searching from the top of the block
BT Thread starts searching from the bottom of the block

START PROCEDURE $EfficientPatternMatchingl(text, NumOfBlocks)$
Divide text into Blocks
Allocate (2* NumOfBlocks) threads (two threads for each block)
for $i = 0$ to $i = NumOfBlocks$ **do**
    Assign block i to FT
    Assign block i to BT
    Start search process at block i, using the specified pattern matching algorithm, with respect to FT
    Start search process at block i, using the specified pattern matching algorithm, with respect to BT
end for
END PROCEDURE $EfficientPatternMatching(text, NumOfBlocks)$

B. working example

Section III-A above explained the proposed EPM approach. This section presents a complete example to clarify how the proposed EPM approach works. Figure 3 presents an example of the searching process using EPM approach. In this example, the text size = 275 words and the target pattern is "compression". The text is divided into three blocks (w = 3). The sizes of the first and second blocks were 92 words, while the third block has 91 words. From the figure we can observe that the target pattern has three occurrences and each occurrence is located at different position. The first occurrence is located at the begging of the text (the first word). The second occurrence is located in the middle of the text (the number of words after it almost equal to the number of the words before it). The last occurrence is located at the end of the text (the last word). Each block is assigned to two threads: (i) a forward thread (FT) and (ii) a backward thread (BT). Forward and backward threads are used to search the pattern concurrently in a timesharing manner from different positions. In detail, the FT scans the text from the top to the bottom of the block. While the BT scans the text from the bottom to the top of the block to speed up the process of finding the required pattern. It is interesting to note that each thread uses a sliding window of size 11 characters, which is equal to the pattern size (compression). With respect to the number of shifts needed to move each sliding window, it depends on the utilised pattern matching algorithm (KMP, FB, and BM). During the searching process, the following will occur:

1) In the first block, $FT1$ will find the pattern before $BT1$.
2) In the second block, the pattern will be found by the fastest between $FT2$ and $BT2$.
3) In the third block, the pattern will be found first by $BT3$.

In the context of evaluation, the execution time, for all threads, was recorded and the minimum time for each pattern occurrence was taken.

IV. Experiments and Evaluation

In this section we present an overview of the adopted experimental set up and the evaluation results obtained. This section is organised as follows: sub-Section IV-A provides an overview of the data collections used to evaluate the proposed Efficient Pattern Matching (EPM) approach and a generic overview of the adopted evaluation measure. While sub-Section IV-B presents and discusses the obtained results.

A. Evaluation Data Collections and Criteria

The suggested Efficient Pattern Matching (EPM) approach was evaluated using three different data collections obtained from "textfiles"[1]. The general characteristics of the data collections are provided in Table I. From the table it can be observed that the evaluation data collections are varied in context of size and topic. In order to evaluate the efficiency of the proposed EPM approach, run time measure was used. Run time refers to the total time an algorithm needs to find each occurrence of the pattern including any preprocessing time. Note that the running times reported in this paper are all in nano second. Since the running time can be affected by many factors such as memory usage and CPU of the system, each test was repeated ten times and all preprocessing times were included. The results were very stable across different runs. All experiments were conducted on Microsoft Windows 7 Professional, 32-bit Operating system with a 3.40 GHz Intel Core i7 processor, and 8 GB memory. Because the performance

[1]a repository that contains copyrighted documents cover a wide range of topics including art, computers, drugs, games, hacking, politics, and many other topics.

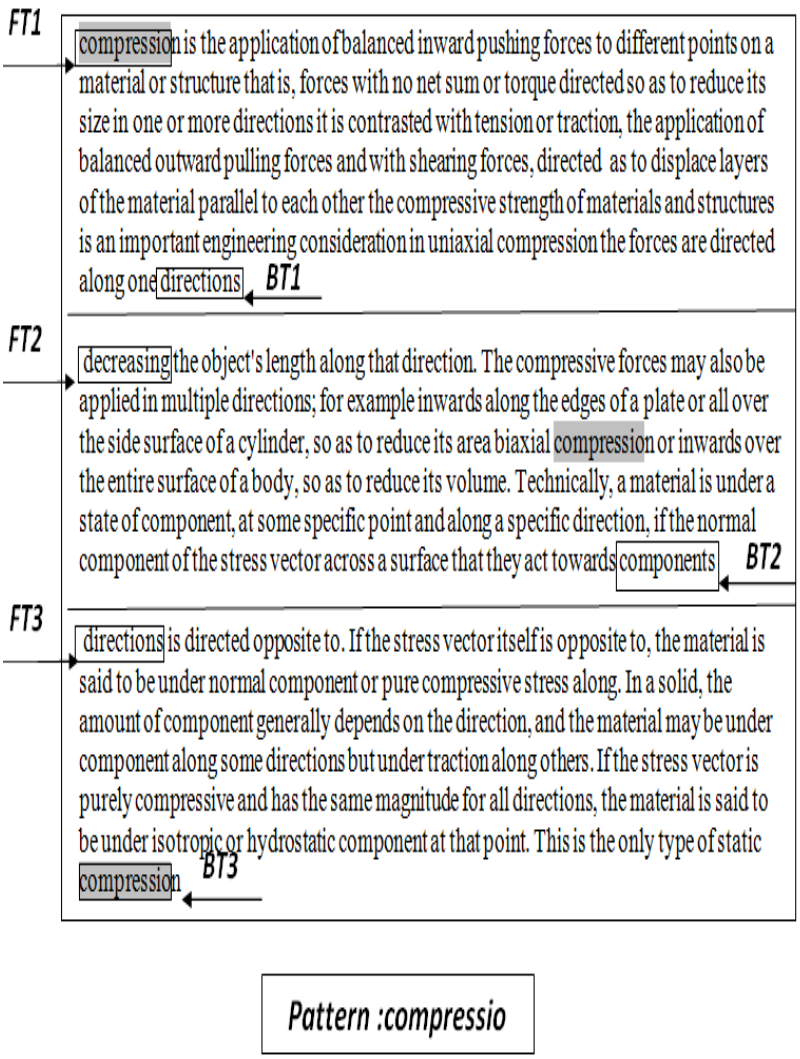

Fig. 3: An example of the searching process using our proposed EPM approach

TABLE I. EVALUATION DATA COLLECTIONS

File Name	Length (KB)	Number of Words	Number of Characters	Description
Abyssal	2KB	355 words	1931 characters	Walkthrough: The Abyssal Zone
Ejournal	239KB	31071 words	262585 characters	Directory of Electronic Journals and Newsletters by Michael Strangelove (July 1992)
Mail-list	1011KB	180403 words	1318707 characters	List of special interest mailing list (June, 14, 1993

of pattern matching process affected by: (i) text length, (ii) pattern length and (iii) pattern occurrence frequency [19], the conducted experiments were designated to take the previous factors into consideration, more specifically:

1) The proposed approach evaluated against three deferent text length (See Table I).
2) The proposed approach evaluated against different pattern length, range from 2 to 26 characters as shown in Table II.
3) The proposed approach evaluated against different pattern occurrence positions: (i) beginning, (ii) middle and (iii) end of the text. Patterns were induced into evaluation texts at these three different positions. Each experiment, reported later in this paper, shows the searching results for all these three positions.

For comparison purposes the three sequential pattern matching algorithms were also applied to the data collections, namely:

1) Brute-Force (BF) sequential exact pattern matching algorithm.

TABLE II. AN EXAMPLE EVALUATION PATTERN WITH VARIABLE LENGTH

Pattern Length	Pattern (P)
2	HO
4	HONO
6	HONOLU
8	HOUNOLULU
10	HOUNOLULULU
12	HOUNOLULULULU
14	HOUNOLULULULULU
16	HOUNOLULULULULULU
18	HOUNOLULULULULULULU
20	HOUNOLULULULULULULULU
22	HOUNOLULULULULULULULULU
24	HOUNOLULULULULULULULULULU
26	HOUNOLULULULULULULULULULULU

2) Knuth-Morris-Pratt (KMP) sequential exact pattern matching algorithm.
3) Boyer-Moore sequential exact pattern matching algorithm.

B. Results and Discussion

In this section, we compare the performance of each sequential algorithm (BF, KMP and Boyer-Moore) with its corresponding multithreading pair (EPM approach) with respect to: (i) different text length, (ii) different pattern length and (iii) different pattern occurrence position. With respect to the EPM approach the text was divided into three blocks ($w = 3$) and each block was assigned two threads running in both directions (i.e forward and backward). Commencing with BF algorithm, Table III presents the obtained results with respect to Abyssal evaluation data collection, Table IV presents the obtained results with respect to Ejournal evaluation data collection and Table V presents the obtained results with respect to Mail-list evaluation data collection.

The tables presents the running time of the forward thread FT, backward thread BT and the minimum time was taken as the fastest result. From the tables it can be observed that the average time taken by the proposed multithreading approach (EPM) is lower than the sequential techniques especially when the pattern is located at the middle or the end of the text. The results are reasonable since the multithreading approaches search the text from multiple sides while the sequential approaches search the text from one side only. This can be justified by the following two advantages of the proposed approach. First, this approach is based on dividing the text into blocks before the search phase. Second, each block is allocated two search threads that scan the block concurrently to match the pattern, with the use of sliding window technique. It is interesting to note that when the pattern occurs at the beginning of the text sequential pattern matching outperforms EPM approach for some cases, the reason behind this is that assigning tasks to threads consumes time in addition to the search time.

Regarding KMP algorithm, Table VI presents the obtained results with respect to Abyssal evaluation data collection, Table VII presents the obtained results with respect to Ejournal evaluation data collection and Table VIII presents the obtained results with respect to Mail-list evaluation data collection. The same as the case of BM algorithm, the multithreading approach outperforms sequential approach in terms of average search time especially when the pattern is located at the middle or the end of the text regardless the pattern length.

With respect to BF algorithm, Table IX presents the obtained results with respect to Abyssal evaluation data collection, Table X presents the obtained results with respect to Ejournal evaluation data collection and Table XI presents the obtained results with respect to Mail-list evaluation data collection. Again, the multithreading approach outperforms sequential approach in terms of average search time especially when the pattern is located at the middle or the end of the text regardless the pattern length.

TABLE III. AVERAGE TIME IN NANO SECONDS FOR SEQUENTIAL BM AND EPM APPROACH FOR BM ALGORITHM (MULTITHREADING BM) WITH RESPECT TO ABYSSAL DATA COLLECTION

Pattern Length	Multithreading BM									Sequential BM		
	Beginning			Middle			End			Begin.	Middle	End
	FT	BT	Min	FT	BT	Min	FT	BT	Min			
2	1841	24962	1841	13885	6550	6550	12677	1570	1570	2203	130943	202602
4	4045	18412	4045	7516	6127	6127	11591	1751	1751	3381	107670	189200
6	3381	13251	3381	7395	6429	6429	11591	2083	2083	3924	91884	167165
8	8331	13704	8331	8029	6973	6973	9206	2143	2143	4105	81892	137674
10	3320	11561	3320	7124	6158	6158	10323	2565	2565	3954	72535	115790
12	3260	12375	3260	7365	6852	6852	8753	2687	2687	4649	67615	104954
14	3652	11017	3652	7154	6490	6490	9417	3079	3079	5222	67826	112077
16	3501	10746	3501	7305	6429	6429	8995	3351	3351	5373	65442	104712
18	4286	11560	4286	7999	5645	5645	8482	3109	3109	5463	66347	102297
20	6459	8965	6459	7154	5674	5674	8180	3290	3290	5162	59133	90585
22	4769	11048	4769	7003	5735	5735	8270	3320	3320	6128	54062	86662
24	5765	9810	5765	6882	6127	6127	7606	3652	3652	5977	57261	103233

TABLE IV. AVERAGE TIME IN NANO SECONDS FOR SEQUENTIAL BM AND EPM APPROACH FOR BM ALGORITHM (MULTITHREADING BM) WITH RESPECT TO EJOURNAL DATA COLLECTION

Pattern Length	Multithreading BM									Sequential BM		
	Beginning			Middle			End			Begin.	Middle	End
	FT	BT	Min	FT	BT	Min	FT	BT	Min			
2	1751	1283135	1751	596201	164233	164233	496502	1208	1208	1177	2147044	2810364
4	2173	1032787	2173	426293	82101	82101	248930	906	906	1902	1532866	1877249
6	1811	727472	1811	482104	54392	54392	164142	966	966	1902	1390180	1625655
8	1841	551196	1841	348659	42922	42922	131242	996	996	1419	1250393	1509019
10	1871	443800	1871	288744	35376	35376	103200	906	906	1721	1293014	1630817
12	1841	362242	1841	245942	28736	28736	89708	1026	1026	2445	1127689	1608781
14	2053	316815	2053	198885	27106	27106	76125	906	906	1600	938670	1509653
16	1992	261366	1992	187415	23846	23846	70269	935	935	1660	854423	1432681
18	2143	246304	2143	167402	19167	19167	64564	1027	1027	2083	951740	1290086
20	2294	220617	2294	151495	19620	19620	60972	996	996	1751	648409	956057
22	2264	199458	2264	130789	16330	16330	56807	905	905	1751	612670	1091226
24	2234	187385	2234	127800	15092	15092	54120	936	936	1902	550065	990709
26	2203	166285	2203	133717	14368	14368	69062	1117	1117	2083	522838	931184

TABLE V. AVERAGE TIME IN NANO SECONDS FOR SEQUENTIAL BM AND EPM APPROACH FOR BM ALGORITHM (MULTITHREADING BM) WITH RESPECT TO MAIL-LIST DATA COLLECTION

Pat. Leng.	Multithreading BM									Sequential BM		
	Beginning			Middle			End			Beg.	Middle	End
	FT	BT	Min	FT	BT	Min	FT	BT	Min			
2	1871	2881654	1871	1283657	1552178	1283657	2066371	845	845	1207	3467394	6982057
4	1871	1815870	1871	518057	759020	518057	1030499	936	936	1388	2061094	3827345
6	1811	1485742	1811	453945	518811	453945	692795	725	725	1268	1909598	3045625
8	2053	1367660	2053	581655	391070	391070	519445	906	906	1449	1687892	2552378
10	2083	1259388	2083	468584	312440	312440	422794	755	755	1479	1554597	2293334
12	2113	1146166	2113	376884	268521	268521	353793	996	996	1600	1483331	2058317
14	2294	1074387	2294	327260	222520	222520	309361	875	875	1660	1421272	1955177
16	2083	1058933	2083	269366	199459	199459	272928	906	906	1660	1388370	1856382
18	2203	1101704	2203	258198	176127	176127	251890	906	906	1721	1357914	1786052
20	3139	918545	3139	226625	157322	157322	227078	872	872	1871	1092501	1454475
22	2294	844261	2294	214310	145097	145097	208967	875	875	1902	1258486	1598063
24	2324	773689	2324	198373	134563	134563	196954	905	905	1902	1519945	1856926
26	2324	765298	2324	191340	125175	125175	184518	875	875	2083	1260297	1555532

TABLE VI. AVERAGE TIME IN NANO SECONDS FOR SEQUENTIAL KMP AND EPM APPROACH FOR KMP ALGORITHM (MULTITHREADING KMP) WITH RESPECT TO ABYSSAL DATA COLLECTION

Pattern Length	Multithreading KMP									Sequential KMP		
	Beginning			Middle			End			Begin.	Middle	End
	FT	BT	Min	FT	BT	Min	FT	BT	Min			
2	1841	25083	1841	13613	6580	6580	22910	1570	1570	1902	59525	95053
4	2324	18473	2324	7456	6128	6128	11681	1811	1811	1660	74376	115579
6	2143	13372	2143	7395	6490	6490	11108	2053	2053	1811	82979	119171
8	2657	13613	2657	7909	6882	6882	9176	2234	2234	1872	71086	108002
10	2777	11742	2777	7063	6128	6128	10323	2566	2566	2505	78874	129283
12	3833	12375	3833	7486	6852	6852	8754	2626	2626	2324	92910	140149
14	3320	11047	3320	7063	6399	6399	9387	3199	3199	2445	78089	129253
16	3652	10565	3652	7274	6278	6278	8995	3230	3230	2234	103324	143379
18	4075	11591	4075	7908	5765	5765	8421	3139	3139	2264	79236	115579
20	5736	10353	5736	7124	5705	5705	8361	3351	3351	2837	93181	141629
22	4739	10927	4739	6973	6580	6580	8180	3230	3230	2868	107851	162728
24	5162	9689	5162	6912	7033	6912	7576	3562	3562	2656	86329	126476
26	4226	8905	4226	7486	5675	5675	8180	3502	3502	3290	79477	130068

TABLE VII. AVERAGE TIME IN NANO SECONDS FOR SEQUENTIAL KMP AND EPM APPROACH FOR KMP ALGORITHM (MULTITHREADING KMP) WITH RESPECT TO EJOURNAL DATA COLLECTION

Pattern Length	Multithreading KMP									Sequential KMP		
	Beginning			Middle			End			Begin.	Middle	End
	FT	BT	Min	FT	BT	Min	FT	BT	Min			
2	1690	1273416	1690	598344	162935	162935	491431	875	875	1147	1999891	3099327
4	1871	1006285	1871	428014	82101	82101	248689	905	905	815	1995484	3095011
6	1811	725510	1811	477787	54392	54392	166648	936	936	996	2004540	3106693
8	1811	546155	1811	342592	42469	42469	127076	966	966	1117	2000495	3160664
10	1932	433205	1932	289287	33716	33716	104257	875	875	1087	2016403	3121845
12	2113	366649	2113	232902	28615	28615	87836	875	875	1268	1935627	3039682
14	2143	325085	2143	219923	24027	24027	77785	875	875	1298	2019331	3128758
16	2053	274315	2053	189165	22035	22035	73257	875	875	1419	1940940	3048586
18	2234	243769	2234	170632	19046	19046	63780	906	906	1449	1912384	3016711
20	2173	221161	2173	152008	17658	17658	60037	996	996	1539	1925092	3043032
22	2113	203594	2113	129823	16179	16179	56958	966	966	1600	1941664	3049884
24	2686	188712	2686	128555	15092	15092	54936	936	936	1751	1919116	3018974
26	2355	169214	2355	119439	14368	14368	51796	1207	1207	1811	1983682	3088401

TABLE VIII. AVERAGE TIME IN NANO SECONDS FOR SEQUENTIAL KMP AND EPM APPROACH FOR KMP ALGORITHM (MULTITHREADING KMP) WITH RESPECT TO MAIL-LIST DATA COLLECTION

Pattern Length	Multithreading KMP									Sequential KMP		
	Beginning			Middle			End			Begin.	Middle	End
	FT	BT	Min	FT	BT	Min	FT	BT	Min			
2	1690	4362730	1690	1028480	3192564	1028480	4248209	1117	1117	1177	4895488	10905149
4	1691	4194933	1691	989209	3072761	989209	4043135	1147	1147	966	4848007	10910461
6	1781	4207581	1781	983142	3057155	983142	3986871	1056	1056	1087	4865725	10938714
8	1992	4253069	1992	1017462	3112574	1017462	4139092	1177	1177	1147	4835752	10850122
10	1962	4267074	1962	1005479	3103821	1005479	4107670	1147	1147	1238	4922865	10951753
12	2023	4214221	2023	983957	3067871	983957	3998885	1087	1087	1238	4925461	10981998
14	2143	4087657	2143	991986	3081635	991986	4047995	1117	1117	1298	4910489	10966121
16	2203	4276251	2203	988666	3046379	988666	4040871	1177	1177	1419	4921446	10967057
18	2234	4171680	2234	990625	3053918	990625	4021814	1027	1027	1509	4892952	10952810
20	2234	4168692	2234	998413	3085611	998413	4064857	1117	1117	1570	4909735	10919064
22	2294	4277265	2294	1008313	3094365	1008313	4031080	1178	1178	1721	4978435	11075450
24	2234	4207690	2234	998081	3065508	998081	4062744	1056	1056	1871	4828598	10837776
26	2505	4166518	2505	993644	3081144	993644	4083964	1086	1086	1902	4944749	11051544

TABLE IX. AVERAGE TIME IN NANO SECONDS FOR SEQUENTIAL BF AND EPM APPROACH FOR BF ALGORITHM (MULTITHREADING BF) WITH RESPECT TO ABYSSAL DATA COLLECTION

Pattern Length	Multithreading BF									Sequential BF		
	Beginning			Middle			End			Begin.	Middle	End
	FT	BT	Min	FT	BT	Min	FT	BT	Min			
2	2264	17447	2264	9599	7788	7788	15394	1358	1358	2354	128226	193607
4	3381	17507	3381	9357	7908	7908	15364	1177	1177	2596	128287	193758
6	2173	19318	2173	9538	8391	8391	15515	1751	1751	2355	114795	196507
8	2234	31875	2234	17084	15032	15032	32418	3682	3682	2807	128771	194153
10	2566	18926	2566	9901	9055	9055	15485	2415	2415	3743	157263	256963
12	3079	18956	3079	10051	9236	9236	15726	2656	2656	4467	187148	323585
14	2445	32750	2445	9870	9387	9387	15635	2747	2747	4799	196022	329561
16	3049	19258	3049	10474	9418	9418	17175	2656	2656	4528	191706	307104
18	30697	32448	30697	16903	9146	9146	15817	2505	2505	4830	187238	305021
20	3200	32116	3200	9931	8874	8874	15817	2324	2324	5192	174229	290804
22	2958	31301	2958	17266	10625	10625	16058	1992	1992	5403	188476	309518
24	2868	30818	2868	17266	14066	14066	15998	1600	1600	7365	242296	412480
26	3139	40960	3139	17779	13402	13402	27951	1660	1660	7094	218902	371488

TABLE X. AVERAGE TIME IN NANO SECONDS FOR SEQUENTIAL BF AND EPM APPROACH FOR BF ALGORITHM (MULTITHREADING BF) WITH RESPECT TO EJOURNAL DATA COLLECTION

Pattern Length	Multithreading BF									Sequential BF		
	Beginning			Middle			End			Begin.	Middle	End
	FT	BT	Min	FT	BT	Min	FT	BT	Min			
2	2113	1038763	2113	480323	242441	242441	709181	453	453	1298	2262986	3207662
4	2355	1027414	2355	469789	238698	238698	707581	392	392	1539	2226311	3177085
6	2294	1039397	2294	478391	238969	238969	705196	634	634	1570	2227578	3177326
8	2324	1028772	2324	469064	240237	240237	706585	966	966	1720	2237570	3184661
10	2234	1030765	2234	472536	238969	238969	714101	876	876	1720	2229088	3181401
12	2596	1026026	2596	468763	300123	300123	709603	875	875	1871	2224198	3180888
14	2505	1040725	2505	469276	245972	245972	707400	936	936	1690	2224892	3166459
16	2445	1028501	2445	469457	240962	240962	710267	996	996	1781	1966507	2816341
18	2475	1026237	2475	469064	239422	239422	707158	936	936	1962	2173004	3120216
20	2596	1043593	2596	468370	239181	239181	707249	815	815	1992	2184836	3131656
22	2475	1027444	2475	469246	238637	238637	707430	845	845	2023	2191447	3140259
24	2626	1040242	2626	477637	239331	239331	708003	724	724	2173	2185983	3134071
26	3653	1037526	3653	470966	238939	238939	705438	453	453	2234	2185832	3136697

TABLE XI. AVERAGE TIME IN NANO SECONDS FOR SEQUENTIAL BF AND EPM APPROACH FOR BF ALGORITHM (MULTITHREADING BF) WITH RESPECT TO MAIL-LIST DATA COLLECTION

Pattern Length	Multithreading BF									Sequential BF		
	Beginning			Middle			End			Begin.	Middle	End
	FT	BT	Min	FT	BT	Min	FT	BT	Min			
2	2204	3300232	2204	730859	2250502	730859	2983505	815	815	1509	4642058	9780652
4	2324	3304186	2324	731554	2246911	731554	2987097	845	845	1539	4695786	9833414
6	2354	3418254	2354	741485	2300941	741485	3033038	1087	1087	1539	4677917	9846816
8	3290	3289426	3290	738255	2256539	738255	3001314	1087	1087	1751	4645680	9829551
10	2807	3293169	2807	730648	2252585	730648	2990719	1177	1177	1720	4710909	9879959
12	2717	3291357	2717	730558	2249235	730558	2992047	1328	1328	1871	4692871	9870812
14	2656	3339140	2656	730648	2257988	730648	2999442	1328	1328	1872	4699439	9883370
16	2596	3294769	2596	730527	2251951	730527	2990327	1208	1208	1962	4678581	9898432
18	2656	3303100	2656	731312	2254910	731312	2984350	1208	1208	2022	4700344	9925145
20	2958	3295402	2958	735568	2249567	735568	2995217	1147	1147	2083	4645710	9793329
22	2837	3380915	2837	732006	2249325	732006	2994613	1147	1147	2234	4676740	9877152
24	2868	3308744	2868	732731	2255936	732731	2997450	966	966	2204	4682113	9847179
26	2958	3305062	2958	731221	2249325	731221	2985557	694	694	2294	4688149	9846484

The previous results are summarized and plotted in the next three figures to compare two approaches: (i) sequential and (ii) multithreading (EPM) with respect to all the three used pattern matching algorithms. Figure 4 presents a comparison between all sequential and multithreading approaches with respect to Abyssal data collection. From the figure it can be observed that: (i) as noted earlier, multithreading approaches clearly outperform sequential approaches in terms of average case time and (ii) the minimum average case time obtained from multithreading BM and multithreading KPM. Figure 5

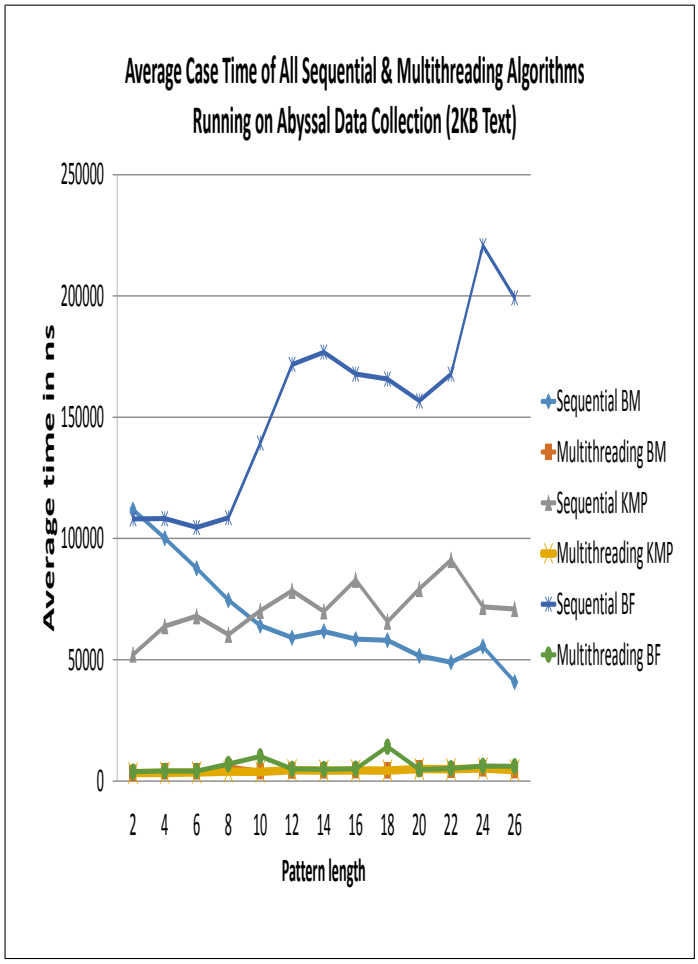

Fig. 4: Comparison between all sequential and multithreading approaches with respect to Abyssal data collection

presents a comparison between all sequential and multithreading approaches with respect to Ejournal data collection. From the figure it can be observed that, and the same as Abyssal data collection, multithreading approaches clearly outperform sequential approaches and the minimum average case time obtained from multithreading BM and multithreading KPM. Figure 6 presents a comparison between all sequential and multithreading approaches with respect to E-mail data collection (the largest data collection). From the figure it can be observed that, and the same as Abyssal and Ejournal data collection, multithreading approaches clearly outperform sequential approaches. However, the minimum average case time obtained from multithreading BM.

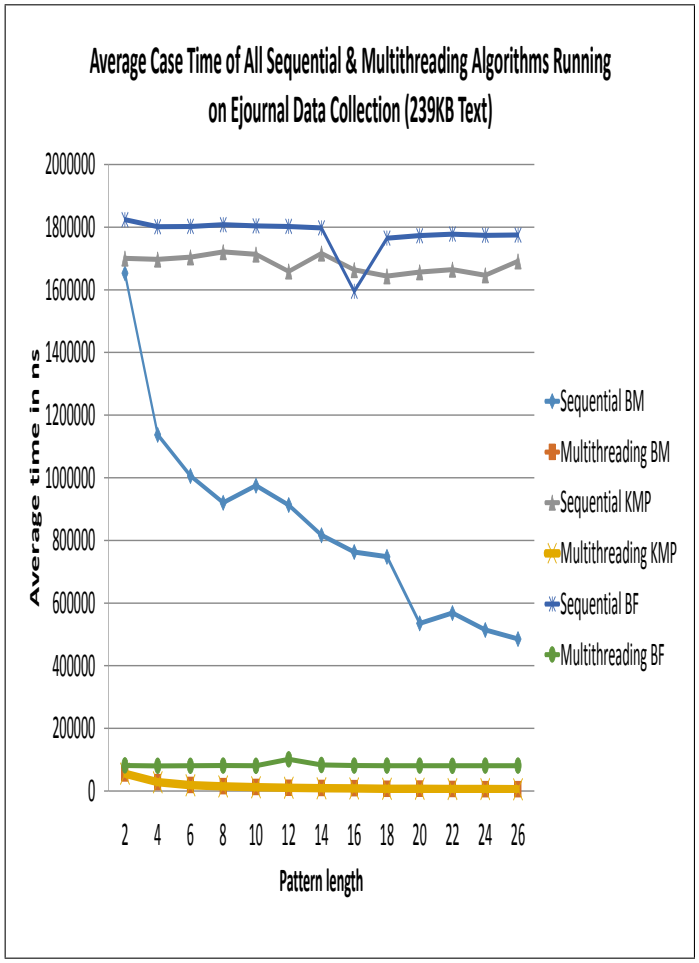

Fig. 5: Comparison between all sequential and multithreading approaches with respect to Ejournal data collection

V. Conclusion and Future Work

In this paper, Efficient Pattern Matching (EPM) approach based on multithreading techniques has been proposed to improve the efficiency of sequential pattern matching algorithms. Three common pattern matching algorithm have been considered: (i) Brute Force, (ii) Knuth-Morris-Pratt and (iii) Boyer-Moore. The central idea was to divide the text into blocks and assign each block to two threads, forward thread and backward thread, to conduct the search process concurrently. The proposed approach was evaluated using different text size and various pattern lengths. From the reported experimental results, presented in this paper, it was demonstrated that the proposed multithreading approach shows remarkable performance gain compared with the traditional sequential approach, especially when the lookup patterns were located at the middle and the end of the text regardless the text length or the pattern length. With respect to future work the authors intend to investigate the effect of multithreading approach on Arabic data collections.

References

[1] C. Charras and T. lecroq, "Exact string matching algorithms-animations in java," URL http://www-igm.univ-mlv.fr/~lecroq/string/index.html, 1997, accessed 1-March-2015.

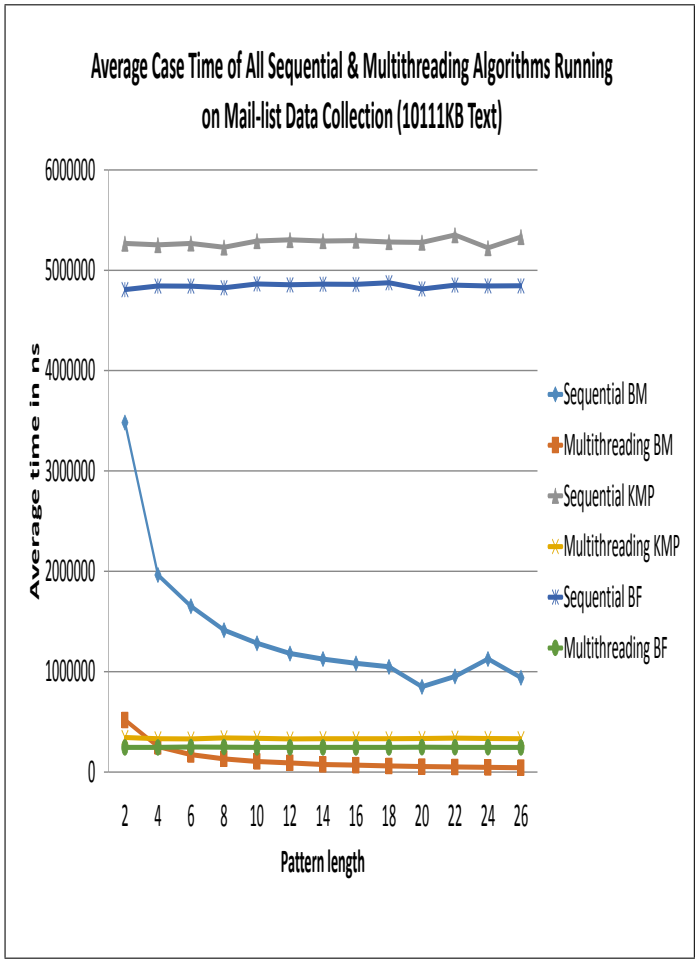

Fig. 6: Comparison between all sequential and multithreading approaches with respect to Mail-list data collection

[2] N. Singla and D. Garg, "String matching algorithms and their applicability in various applications," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 1, no. 6, pp. 218 – 222, 2012.

[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.

[4] C. Charras and T. Lecroq, *Handbook of Exact String Matching Algorithms*. King's College Publications, 2004.

[5] A. Levitin, *Introduction to the Design and Analysis of Algorithms (2Nd Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2006.

[6] D. Knuth, J. Morris, and V. Pratt, "Fast pattern matching in strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323–350, 1977.

[7] J. Morris and V. Pratt, "A Linear Pattern Matching Algorithm," Computing Center, University of California, Berkeley, Tech. Rep. 40, 1970.

[8] R. Boyer and J. Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762–772, Oct. 1977.

[9] S. Faro and T. Lecroq, "The exact online string matching problem: A review of the most recent results," *ACM Comput. Surv.*, vol. 45, no. 2, pp. 13:1–13:42, 2013.

[10] A. Hudaib, R. Al-Khalid, D. Suleiman, M. Itriq, and A. Al-Anani, "A fast string matching algorithm with two sliding windows (tsw)," *Journal of Computer Science*, vol. 4, no. 5, pp. 393–401, 2008.

[11] S. Faro and T. Lecroq, "A multiple sliding windows approach to speed up string matching algorithms," in *Experimental Algorithms - 11th International Symposium, SEA 2012, Bordeaux, France, June 7-9, 2012. Proceedings*, 2012, pp. 172–183.

[12] A. Hudaib, R. Al-Khalid, A. Al-Anani, M. Itriq, and D. Suleiman, "Four sliding windows pattern matching algorithm (fsw)," *Journal of Software Engineering and Applications*, vol. 8, no. 3, pp. 154–165, 2015.

[13] M. MacDonald, *Pro .NET 2.0 Windows Forms and Custom Controls in VB 2005*. Apress, 2007.

[14] H. Deitel and P. Deitel, *Java SE8 for Programmers*, 3rd ed. Prentice Hall Press, 2014.

[15] T. Ungerer, B. Robič, and J. Šilc, "A survey of processors with explicit multithreading," *ACM Comput. Surv.*, vol. 35, no. 1, pp. 29–63, 2003.

[16] N. Kofahi and A. Abusalama, "A framework for distributed string matching based on multithreading," *The International Arab Journal of Information Technology*, vol. 9, no. 1, pp. 30–38, 2012.

[17] S. N. Devi and S. P. Rajagopalan, "An index based pattern matching using multithreading," *International Journal of Computer Applications*, vol. 50, no. 6, pp. 13–17, 2012.

[18] A. Rasool, N. Khare, H. Arora, A. Varshney, and G. Kumar, "Multithreaded implementation of hybrid string matching algorithm," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 4, no. 3, pp. 438–441, 2012.

[19] W. Smyth, S. Wang, and M. Yu, "An adaptive hybrid pattern-matching algorithm on indeterminate strings," in *Proceedings of the Prague Stringology Conference 2008*, J. Holub and J. Žďárek, Eds., Czech Technical University in Prague, Czech Republic, 2008, pp. 95–107.