Parallel Enhanced Pattern Matching Algorithm with Two Sliding Windows PETSW

Wafa Dababat
SE Department,
Balqa' Applied University
Salt-Jordan

Mariam Itriq
BIT Department
Jordan University
Amman-Jordan

ABSTRACT
String matching problem is one of the most essential problems in many computer science fields, such as DNA analysis, artificial intelligence, internet search engines and information retrieval. Today, the speed and performance of string matching algorithms is critical and must be improved to meet recent developments in hardware processing environments. The improvement in performance gained by the use of a multi core processor depends very much on the software algorithms used and their implementation. However, the most important factor when writing a parallel algorithm is the fraction of the algorithm that can run simultaneously on multiple cores. In this paper, an efficient algorithm for string matching, Enhanced Pattern Matching Algorithm with Two Sliding Windows (ETWS), is adapted to be implemented under a real parallel environment (PETWS), to enhance the performance of the sequential algorithm through providing less execution time to make it more suitable for today's applications.

General Terms
String Matching, Pattern Matching, Parallel Algorithms.

Keywords
Multi-Core Computer Algorithms, ETSW String Matching Algorithm, Data Partitioning Parallel Approach.

1. INTRODUCTION
In the last two decades, string matching algorithms serve as a kernel module in various computer applications and fields, such as DNA analysis, web based applications with search facility, social networks, dictionaries, antivirus and anti-spam software's. All these applications maintain a large amount of data and involve deferent types of alphabets and many of them are mobile applications requiring a good performance and response time. Therefore, several efficient algorithms have been introduced to both parallel and single processor environments. The size of documents to search in is rapidly increasing, so we need to concentrate on parallel and distributed environments to cope with these changes.

Adapting efficient single processor Pattern Matching Algorithms for parallel and distributed environments is an option that has been applied by several researchers. This approach is followed in this paper. A time efficient pattern matching algorithm Enhanced Pattern Matching Algorithm with Two Sliding Windows (ETWS) [7] is analyzed and applied to multi-processor computer environments to achieve a speedup on the algorithms operations.

The rest of this paper is organized as follows. Section 2 gives the review of several algorithms. Enhanced Pattern Matching Algorithm with Two Sliding Windows (ETWS) is discussed in section 3. In Section 4, describe the proposed Parallel Enhanced Pattern Matching Algorithm with Two Sliding Windows (PETSW) in details. In section 5, the experiment results of comparisons between the proposed algorithm (PETSW) on different multi core computers and the original one (ETWS) are given. And in section 6 conclusions and future work are discussed.

2. RELATED WORKS
Many studies on string matching problem have been conducted over the years to develop efficient algorithms; most of these algorithms enhance number of attempts and number of comparisons through using sliding window(s) with the best shift values [3][4][7-13].

A number of approaches for parallel processing exist, message passing approach, data partitioning approach, shared data approach and others. The main idea behind the parallel processing is to divide a heavy task into sub-tasks that can be solved simultaneously through using multi-core computers or more than one thread to enhance the overall execution time of the computations. In addition to the importance of parallel string matching algorithms in many areas such as biological applications [2], search engines applications [19], intrusion detection in network applications [1][11] and many other applications, several parallel string matching algorithms have been proposed in different platforms such as Graphics Processing Unit (GPU) [14][18], Field-Programmable Gate Array (FPGA) [15] and a multi-core architecture with message passing interface [16][17].

In this paper, a well known algorithm (ETSW) is selected and data partitioning approach is used to adapt the algorithm in real parallel multi-core environment.

3. ENHANCED PATTERN MATCHING ALGORITHM WITH TWO SLIDING WINDOWS (ETSW)
The Enhanced Two Sliding Windows algorithm (ETSW) [7] is an extended version of Two Sliding Window algorithm (TSW)[4] as well as ERS-A[3], FSW[9], EBR[5], EERS-A[10] and DBR[8] algorithms.

In general ETSW [7] in order to improve the search process it scans the text as well as the pattern from both sides simultaneously. The ETSW algorithm uses two sliding windows to search the text from both sides these windows are slides in parallel. Comparisons done with the pattern is also done from both sides simultaneously. The length of each window is m which is the same length as the pattern. The text is divided into left and right parts, and the pattern is also divided into left and right parts. Each part of the text is of length $\lceil n/2 \rceil$ while each part of the pattern is of length $\lceil m/2 \rceil$. ETSW algorithm stops when one of the two sliding windows finds the pattern or the pattern is not found within the text string at all.

ETSW algorithm utilize BR bad character shift function as the TSW algorithm [4] done to get better shift values during the searching process.

ETSW perform two main phases, the pre-processing and searching phase.

3.1 Pre-processing phase

In this phase ETSW generate two arrays of one-dimensional named nextl, nextr. The shift values of the nextl array are calculated according to Berry-Ravindran bad character algorithm (BR) [6] as in equation (1). The shift values needed to search the text from the left side. The shift values of the nextr array that are needed to search the text from the right side are calculated according to the TSW shift function as in equation (2). During the searching process, the nextl and the nextr arrays will be invariable [7].

$$BadChar\ shiftl[a,b] = min\begin{Bmatrix} 1 & if\ p[m-1]=a \\ m-i & if\ p[i]p[i+1]=ab \\ m+1 & if\ p[0]=b \\ m+2 & Otherwise \end{Bmatrix} \quad (1)$$

$$BadChar\ shiftr[a,b] = min\begin{Bmatrix} m+1 & if\ p[m-1]=a \\ m-((m-2)-i) & if\ p[i]p[i+1]=ab \\ 1 & if\ p[0]=b \\ m+2 & Otherwise \end{Bmatrix} \quad (2)$$

3.2 Searching phase

In this phase the ETSW algorithm beginning the matching process from two directions from left to right and from right to left. In mismatch cases, during the searching process from the left, the left window is shifted to the right, while during the searching process from the right; the right window is shifted to the left. Both windows are shifted according to shift values calculated for both sides in pre-processing phase until the pattern is found or the windows reach the middle of the text [7]. ETSW algorithm [7] finds either the first occurrence of the pattern in the text through the left window or the last occurrence of the pattern through the right window. Fig.1 shows the main operation in this phase [7].

```
L=m-1; //text index used from left
R=n-(m-1)-1; //text index used from right
Tindex=0;//text index used to control the scanning process
While (Tindex<=⌈n/2⌉)
Begin
  foundleft = false;
  foundright = false;
  l=m-2 ;  // pattern index used at left side from the end of the pattern
  r=0;  // pattern index used at right side from the beginning of the pattern
  temp-lindex=temp-rindex=0;//keep record of the text index where the pattern match the text during comparison
  temp_newlindex=0; // pattern index used at left side from the beginning of the pattern
  temp_newrindex= (m-1); // pattern index used at right side from the end of the pattern
 if (P[m-1]=T[L]  and  p[0]=T[L-m+1])
    begin
       temp-lindex=L;
      L=L-1;
      temp_newlindex++;
     while      ((l>=0      and      P[l]=T[L])      and (P[temp_newlindex]=T[L-l+ temp_newlindex] ))
         { L=L-1, l=l-1;  temp_newlindex++;
           if  ((L-l+ temp_newlindex) >=L)
           {foundleft = true; exit from while loop;}
          }   //search from left
    end
 if (P[0]=T[R] and  p[temp_newrindex]=T[L+m-1])
  begin
    temp-rindex=R;
   R=R+1;
   temp_newrindex--;
   while(      (r<m      and      P[r]=T[R])      and P[temp_newrindex]=T[R+ temp_newrindex-r]  )
      { R=R+1, r=r+1; temp_newrindex --;
          if  (R+ temp_newrindex-r<=R)
           {foundright = true; exit from while loop;}
            }   //while
       }  //search from right
  end
   if (foundright) {display "match at right:  "+ temp-rindex) ; exit from outer loop;}
   if (foundleft) {display "match at left:  "+ temp-lindex –m +1); exit from outer loop;}
   //exit in case if we search for one occurrence the first or last one
   R= temp-rindex; //to avoid skipping characters after partial matching at right
   L=temp-lindex; // to avoid skipping characters after partial matching at left
   if(not foundleft and not foundright){ display ("not found"); exit from outer loop;}
   L=L+get(shiftl);//from pre-processing step
   R=R-get(shiftr);//from pre-processing step
   Tindex= Tindex+1;
   End;
```

Fig.1 :ETSW Algorithm [7]

4. THE PROPOSED PARALLEL ENHANCED PATTERN MATCHING ALGORITHM WITH TWO SLIDING WINDOWS (PETSW)

PETSW algorithm implements ETSW pattern matching algorithm in a parallel environment. According to the analysis of original ETSW algorithm in previous section, the expensive phase of the algorithm is the searching phase in which the characters of the pattern matches the characters of the text window [7]. To reduce the cost of this phase, it is carried out in PETSW algorithm using multiple simultaneous threads in multi-core environment. The pre-processing phase remains as it is because the shift values in nextl, nextr arrays are invariable as considered in previous section.

The Parallel Enhanced Pattern Matching Algorithm with Two Sliding Windows (PETSW) through a master thread perform the pre-processing phase and then distribute the works to the worker (slaves) threads to perform the searching phase in parallel, finally the master thread collects and displays the results.

While the ETSW algorithm finds either the first occurrence of the pattern in the text through the left window or the last occurrence of the pattern through the right window as mentioned in previous section the PETSW algorithm finds more results since each thread will find a result in its part of the text (e.g. two threads will give us two results, three threads will give us three different results and so..).

PETSW algorithm is applied on two different hardware environments: the first computer has two CPUs and the second one has eight CPUs.

The PETSW algorithm is divided into three phases, Fig.2 illustrates the PETSW phases:

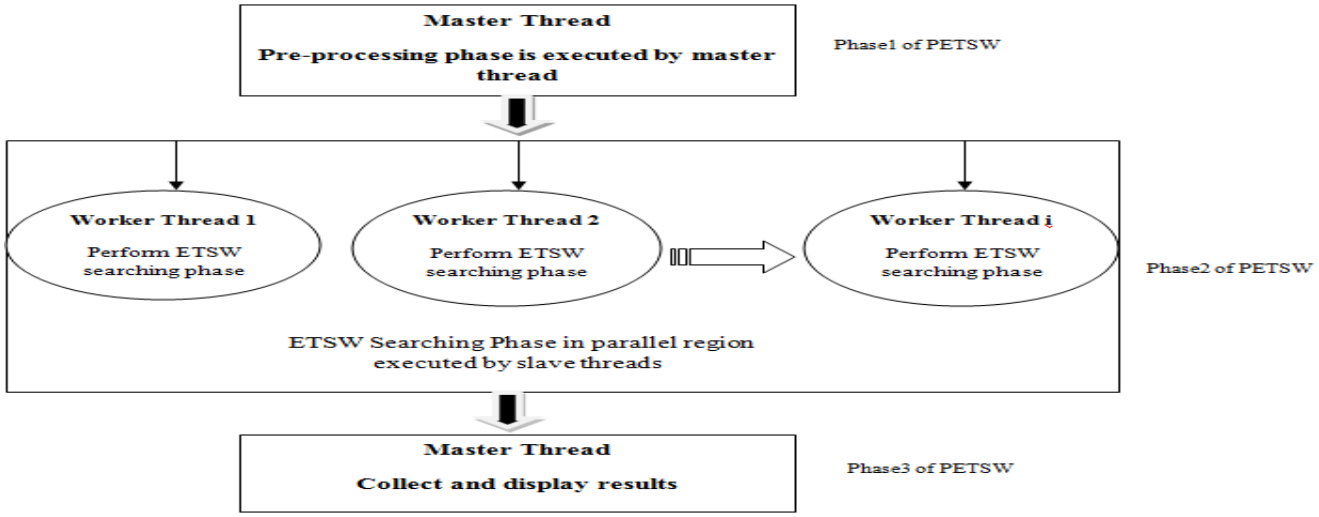

Fig.2 :PETSW main phases

4.1 Phase1: Pre-processing

In this phase the server make three main operations:

1. The input text string of size n is sliced into 'i' subtexts such that each text partition holds ((n/S)+m-1) text string characters with m-1 text characters overlapping in each partition, here S refers to the number of threads in the current run, m and n being the lengths of text and pattern string respectively .The number of sub texts obtained after partitioning the text string using the above formula equals the number of threads in the program, i.e., i=S, thereby representing the static allocation of the threads. Fig.3 shows this part.

Input : *the file contain the text of size n , the number of threads S, and the size of the pattern m*

Output *: S sub texts, each t_i assign to S_i*

1. *Begin*
2. *c=0*
3. *While (c<n/m+m and not EndOfFile) Do*
4. *$t_i=t_i+nextChar$*
5. *c=c+1*
6. *End While*
7. *Assign t_i to S_i*
8. *Repeat steps 2 to 7 for each thread $S_i \in S_{i=1,.....S}$*
9. *End*

Fig.3 :Text partitioning phase

Below an example is presented to clarify the partitioning step using a text with 80 characters and a pattern with 5 characters.

Input:
Text =*"Copyright laws are changing all over the world. Be sure to check the copyright l", n=80*
Pattern (P) = "world" ,m=5

Assume we have 4 threads (S=4), then the number of subtexts i=S=4

Output:

S1 subtext is: "Copyright laws are chang"

S2 subtext is: "hanging all over the wor"

S3 subtext is: "world. Be sure to check"

S4 subtext is: "heck the copyright l"

2. The server generates the two arrays nextl and nextr which contain the shift values (the pre-processing phase in ETSW) as shown in Fig.4.

3. The server distributes the i sub text partitions and the nextl and nextr arrays on S threads in the program such that thread Sj will get one text partition Tj where 1<=j<=S and receive a complete nextl and nextr.

Input : *the file contain the text of size n and the pattern of size m*

Output *:nextl, nextr arrays that contain shift values for both sliding window*

1. *Begin*
2. *shiftl=shiftr=m+2*
3. *for (each character* $p_i \in P_{i=0,\ldots..m-2}$ *)*
4. *{nextl[i]=m-i,nextr[i]=m-((m-2)-i)}*
5. *if P[m-1]=a {shiftl=1}*
6. *else if p[0]=b { shiftl=m+1}*
7. *if p[0]=b { shiftr=1}*
8. *else if P[m-1]=a{shiftr=m+1}*
9. *End*

Fig.4 :The pre-processing in ETSW algorithm

4.2 Phase2: Parallel Computation and searching

All threads will compute use ETSW simultaneously and returns the result back to the server.

4.3 Phase3: Computing Results

The server will process the results returned from the threads to compute the final result.

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1 Experimental Environment and Results

Our experiment was conducted using java Netbeans 8.1 on 2 core machines with 2.2 GHz, 6 GB RAM in first shot and 8 core machine with 3.4 GHz, 8 GB RAM in second shot. To obtain the results we have applied PETSW algorithm on different data files and patterns. In this work we will show the results of the experiment on a 24MB text file that contains 3562345 words of random text content. We applied the algorithm on different patterns and took the average for each run.

Table 1 contains the running time of running the PETSW algorithm on two core computer. Table 2 shows the speedup obtained by running the algorithm on two core computer. Fig.5 diagrams the running time obtained on two core computer and Fig.6 speedup achieved by the algorithm on the two core computer. Table 3 holds the running time computed for the algorithm on 8 core computer. Table 4 shows the speedup obtained by this experiment on 8 core computer. Fig.7 shows the dropdown of the running time using the algorithm on 8 core computer and Fig.8 presents the speedup gained by running the algorithm on the 8 core machine.

Table 1: Running time of Parallel ETSW on two core processor

Number of threads	Time(ns) average for multiple patterns
1	90181647
2	50513741
3	59549913
4	54096205
5	59854089

Table 1 contains the results of applying the parallel algorithm to two core processor. Each time (ns) represents the average time of the algorithm using several multiple different patterns, the first row shows the result of the sequential algorithm, at each experiment the number of threads was increased by one and the average time was recorded. The lowest time taken was 50513741 ns, when having two threads.

Table 2: Speedup of Parallel ETSW on two core processor

Number of threads	Speedup
1	1
2	1.785289413
3	1.514387553
4	1.667060508
5	1.506691498

Table 2 shows the speedup obtained by comparing the sequential time in Table.1 using the speedup formula in equation (3).

$$Speedup = \frac{running\ time\ of\ the\ sequentail\ algorithm}{running\ time\ of\ the\ parallel\ algorithm} \quad (3)$$

The maximum speedup achieved is 1.785289413 when having two threads.

Table 3 contains the results of applying the parallel algorithm to eight core processor. Each time (ns) represents the average time of the algorithm using several multiple different patterns, the first row shows the result of the sequential algorithm, and at each experiment the number of threads is increased by one and the average time is recorded. The lowest time taken was 18380920 ns, when having eight threads.

Table 3: Running time of Parallel ETSW on 8 core processor

Number of threads	Time(ns) average for multiple patterns
1	80073014
2	42331342
3	31202554
4	28331868
5	25985803
6	24369878
7	20711061
8	18380920
9	21893108
10	23858479

Table 4 :Speedup of Parallel ETSW on 8 core processor

Number of threads	Speedup
1	1
2	1.891577498
3	2.566232687
4	2.826252544
5	3.081413878
6	3.285737171
7	3.866195653
8	4.356311545
9	3.657453021
10	3.356165915

Table 4 shows the speedup obtained by comparing the sequential time in Table.3 to the time obtained by the parallel algorithm on different number of threads using equation (3). The maximum speedup obtained is 4.356311545 when the number of threads was eight.

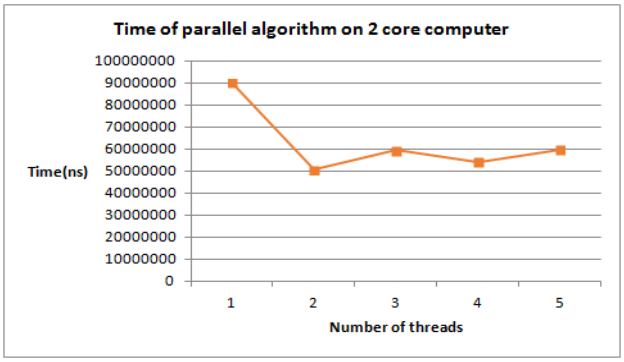

Fig.5 :Running time of Parallel ETSW on two core processor

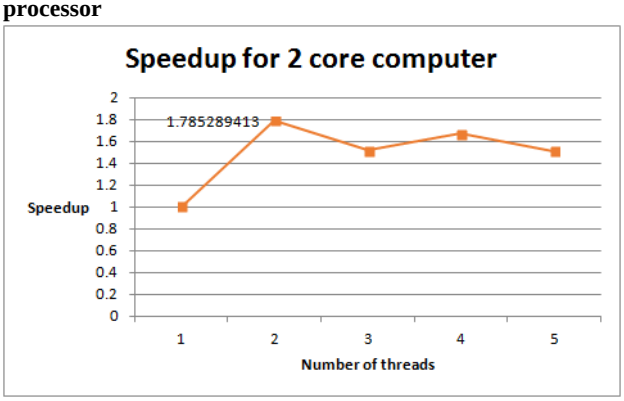

Fig.6 :Speedup of Parallel ETSW on two core processor

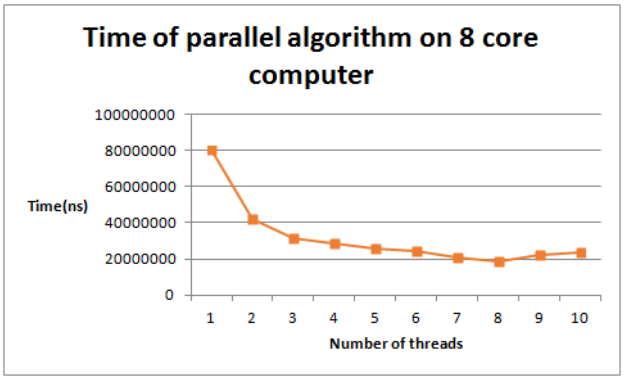

Fig.7 :Running time of Parallel ETSW on 8 core processor

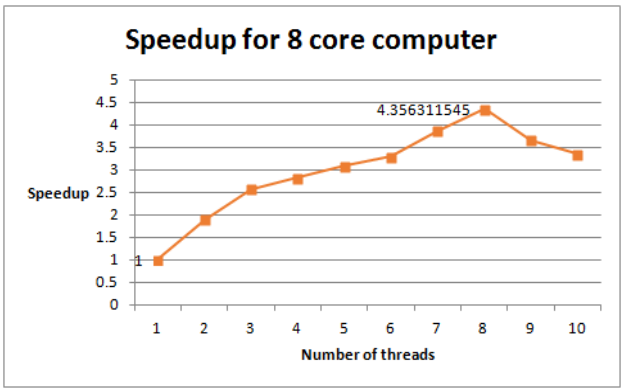

Fig.8 :Speedup of Parallel ETSW on 8 core processor

5.2 Experimental Results Analysis

Using the first machine; that has two CPUs. The algorithm was implemented as one thread to obtain a sequential time result, then the number of threads was increased by one at a time, the experiment was stopped on 5 threads because no extra speedup seemed to be obtained. The execution time has a lowest value when the number of threads is 2, that is equal to the number of CPUs in the computer. The maximum speedup is about 1.8, which is also achieved when having two threads.

The speedup obtained when using three threads and more was decreasing rather than increasing because the time spent on communication between threads seems to overcome the desired increase in performance. As a result, the best time and best speedup will be when number of parallel processes equals the number of CPUs in the machine.

When the second computer having 8 CPUs was used, it is also started with a sequential run of the algorithm, then the number of threads was increased one at a time, which means that one more CPU is used at each run. The running time of the algorithm was decreasing each time we added a new thread, hitting the lowest run time when having 8 threads, when the 9th thread was added, the time increased instead of decreasing, so indicating that no extra time saving may be obtained by introducing new threads, so the experiment stopped after running 10 threads.

Regarding speedup, the best speedup obtained was 4.35 when the number of processes was eight. No better speedup was obtained on this machine.

So, it is noticed that the parallel pattern matching algorithm will continue to give good results until number of threads exceeds the number of CPUs in the machine, in this case it will begin to slow down rather than speeding.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this work, the design and evaluation of a parallel version of the ETSW pattern matching algorithm using text partitioning approach is presented. The key idea of the proposed algorithm is based on the use of multi-core processing system to optimize the performance through reducing search time by an accepted factor compared to the original ETSW algorithm. The experimental results show that the running time and speedup of the proposed algorithm are much better than the sequential approach.

6.2 Future Work

The efficiency of the PETSW algorithm presented in this paper can be tested on search engines and compared to other string matching algorithms. Also, the algorithm may be tested on mobile environments so that battery consumption may be estimated.

7. REFERENCES

[1] CHAO Y. 2012. An Improved BM Pattern Matching Algorithm in Intrusion Detection System. Applied Mechanics and Materials, vol. 148 – 149, 1145-1148.

[2] SENAPATI K.K., MAL S. & SAHOO G. 2012. RS-A Fast Pattern Matching Algorithm for Bio-logical Sequences. International Journal of Engineering and Innovative Technology (IJEIT), 1(3), 116- 118.

[3] SULEIMAN D,HUDAIB A, AL-ANANI A,AL-KHALID R & ITRIQ M, 2013. ERS-A Algorithm for Pattern Matching. Middle East Journal of Scientific Research, 15(7), 1067-1075.

[4] HUDAIB A., AL-KHALID R., SULEIMAN D., ITRIQ M. & AL-ANANI A, 2008. A Fast Pattern Matching Algorithm with Two Sliding Windows (TSW). Journal of Computer Science, 4(5), 393-401.

[5] SULEIMAN D, 2014. Enhanced Berry Ravindran Pattern Matching Algorithm (EBR). Life Science Journal, 11(7), 395- 402.

[6] BERRY, T. & RAVINDRAN, S., 2001. A Fast String Matching Algorithm and Experimental Results. In Proceedings of the Prague Stringology Club Workshop ’99 (eds Holub, J.and Simanek, M), Collaborative Report DC-99-05, Czech Technical University, Prague, Czech Republic, 16-26.

[7] ITRIQ M., HUDAIB A., AL-ANANI A., AL-KHALID R. & SULEIMAN D, 2012. Enhanced Two Sliding Windows Algorithm for Pattern Matching (ETSW). Journal of American Science, 8(5), 607- 616.

[8] Hudaib A., Suleiman D. & Awajan A, (2016, April). Dynamic Berry Ravindran Algorithm for Pattern Matching (DBR), 6th International Conference on Applied Computer Science (ACS '16), At Istanbul, Turkey, (pp 15-17).

[9] HUDAIB A., Al-KALID R., AL-ANANI A, ITRIQ M & SULEIMAN D, 2015. Four Sliding Windows Pattern Matching Algorithm (FSW). Journal of Software Engineering and Applications, 8, 154-165.

[10] SULEIMAN D., ITRIQ M., AL-ANANI A., Al-KHALID R. & HUDAIB A, 2015. Enhancing ERS-A Algorithm for Pattern Matching (EERS-A). Journal of Software Engineering and Applications, 8, 143-153.

[11] Naik M. S., & Geethanjali N. 2015. Performance Study of the Running Times of well known Pattern Matching Algorithms for Signature-based Intrusion Detection Systems, International Journal on Recent and Innovation Trends in Computing and Communication, 3(6). 4177–4180

[12] Hudaib A., Suleiman D. & Awajan A., 2016. A Fast Pattern Matching Algorithm Using Changing Consecutive Characters, Journal of Software Engineering and Applications 9,399-411.

[13] Faro, S. & Lecroq, T. 2012. A Multiple Sliding Windows Approach to Speed Up String Matching Algorithms. SEA, 172-183

[14] Xu D., Zhang H. & Fan Y. 2013. The GPU-based high-performance pattern-matching algorithm for intrusion detection, Journal of Computational Information Systems, 9, 3791-3800.

[15] Kim H. J. 2015. A failureless pipelined Aho-Corasick algorithm for FPGA-based parallel string matching engine, Lecture Notes in Electrical Engineering, 339, 157-164.

[16] Qu J., Zhang G., Fang Z. & Liu J. 2016. A Parallel Algorithm of String Matching Based on Message Passing Interface for Multicore Processors, International Journal of Hybrid Information Technology, 9(3), 31-38.

[17] Liu J., Li F. & Sun G. 2016. A Parallel Algorithm of Multiple String Matching Based on Set-Partition in Multi-core Architecture, International Journal of Security and Its Applications, 10(4), 267-278.

[18] Lin C., Wang G. & Huang C. 2014. Hierarchical parallelism of bit-parallel algorithm for approximate string matching on GPUs, Computer Applications and Communications (SCAC), IEEE Symposium on. IEEE, 76–81.

[19] Singla N. & Garg D. 2012. String Matching Algorithms and their Applicability in various Applications ,International Journal of Soft Computing and Engineering (IJSCE) 1(6).