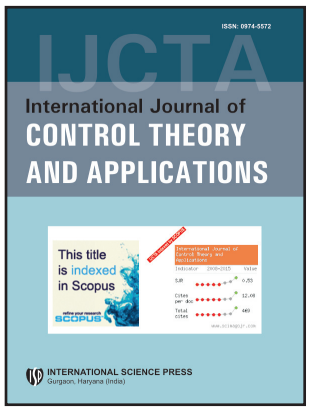

International Journal of Control Theory and Applications

ISSN : 0974-5572

Volume 10 • Number 24 • 2017

A Fork Join Frame Work for Brute Force, Boyer Moore and Skip Search String Matching Algorithms

S. Viswanadha Raju[a], K.K.V.V.V.S. Reddy[b] and Chinta Someswara Rao[c]

[a]*Department of CSE, JNTUHCEJ, JNTUniversity Hyderabad, Telangana, India*
[b]*Research Scholar, Rayalasheema University, AP, India*
[c]*Department of CSE, SRKR Engineering College, Bhimavaram, AP, India*

Abstract: In present days information retrieval play the vital role in every one daily life. If retrieval system returns irrelevant information that lead to wastage of time and effort. For relevant information retrieval different researchers proposed retrieval algorithms, but still there is a necessity of these algorithms. For this purpose in this paper, we take Brute force, Boyer moore and Skip search retrieval algorithms from literature and converted them into parallel algorithms by adopting fork join concept. The parallel algorithms returns most relevant results as well as reduces the search time.

Keywords: Information retrieval; Parallel, fork join; Brute force; Boyer moore; Skip search.

1. INTRODUCTION

There is a necessity for the improvement of the performance of searching on the current day text collections because of the exponential growth of textual databases with the time[1-6]. In retrieving the data from these textual databases, special purpose algorithms have been developed by the researchers, those are parallelizing the comparison[1-6]. But these algorithms are not full fill the users requirement, so still there is a scope of necessity to develop a text retrieval algorithms. For this purpose in this study, Brute force[7], Boyer moore[8] and Skip search[9] string matching algorithms are considered and convert them into parallel algorithms. In these parallel algorithms a number of comparisons will be performed parallel and reduced the search time.

2. LITERATURE SURVEY

This section three different string matching approaches Brute force, Boyer moore and Skip search that are basic to this study are discussed.

Brute force[7]: A basic algorithm in the study of string matching problem is the Brute Force (BF) algorithm. When the length of the text is n and the length of the pattern is m. The comparison will be performed one by

one character from left to right. If a mismatch occurs, the sliding windows get shifted one position to the left and restarts to match from the first position of the pattern.

Boyer moore[8]: This method developed by Boyer Moore (BM) algorithm in 1977. In search process, the character comparisons are carried out form right to left in the pattern. When a mismatch occurs in order to reduce the number of comparisons, this method follows two rules called bad character rule and good suffix rule.

Skip search[9]: This approach is proposed by Christian et. al.,. It has two phases called pre processing and searching. In pre processing, builds the buckets that contain information regarding all the alphabets position in the pattern. These alphabets start recording from the first left position of the pattern. In searching compare the character of pattern with the text with the basis of bucket information.

3. METHODOLOGY

The multi pattern multi-processor parallel string matching algorithm reads the directory and pattern set. Reads one file from the directory, opens the file, reads the file line by line, appends the line to string buffer. In this study three string matching algorithms brute force, boyer moore and skip search are considered. The fork join mechanism is adopted from java technology and this concept is applied on brute force, boyer moore and skip search string matching algorithms, and converted into parallel algorithms those are renamed as parallel brute force, boyer moore and skip search string matching algorithms.. These algorithms are actual process is given in algorithm 1, 2 and 3 respectively. Algorithm 1 give the actual fork join process of brute force approach. Algorithm2 give the actual fork join process of boyer moore approach, but in this approach fork_join functioning is similar as in Algorithm 1. Algorithm 3 give the actual fork join process of skip search approach, in this algorithms also fork_join functioning is similar as in Algorithm 1.

Algorithm 1: Brute Force String Matching Algorithm With Fork Join

Input: Monkey chromosome files, let these are considered as *Text files 'T' of length 'n' and Patterns (P_1, P_2, ...) of length m*

Output: *The number of occurrences and the positions of the patterns (P_1 , P_2, ...)*

```
/* Main */
1    n←T.length, m←P1.length or P2.length or ...or Pn.length
2    for each P1, P2,...Pn
3    for i ← 0 to n-m do
4    begin
5            fork_join(T);
6            count←search_process(T,P,i,count);
7    end for
8    end for
/* Search */
9    int search_process(Char[] T, Char[] P, int i, int count)
10   begin
```

```
11              j′← P.length;
12              j″← 0;
13              while (j′>=0 && T[i - j′] == P[j″])
14              do
15                      j′←j′-1;
16                      j″←j″-1;
17              done;
18              if (j′== -1)
19                      count++;
20              end if
21      return count;
22      end search_process;
/* fork_join */
23      public static void fork_join(Char[] T, Char[] P)
24      begin
25      n←T.length;m←P.length
26      nc←AvailableProcessors()
27      start_pos←starting position of the text i.e.0
28      end_pos←n;
29      split_pos ← n / 2^nc
30      if (end_pos - start_pos > split_pos)
31      begin
32      mid_pos = (start_pos + end_pos) / 2 ;
33      task_assignment=invoke(asList(new search_process (T, start_pos, mid_pos+m-1,P), new search_
        process (T, mid_pos, end_pos,P)));
34      end;
35      invoke(all_tasks);
36      end fork_join;
```

Algorithm 2: Boyer Moore String Matching Algorithm With Fork Join

Input: Monkey chromosome files, let these are considered as *Text files 'T' of length 'n' and Patterns (P_1, P_2, ...) of length m*

Output: The number of occurrences and the positions of the patterns (P_1 , P_2, …)

```
/* Main */
1    n←T.length, m←P1.length or P2.length or ...or Pn.length
2    for each P1, P2,...Pn
3    for i ← 0 to n-m do
4    begin
5            fork_join(T);
6            count←search_process(T,P,i,count);
7    end for
8    end for
/* Search */
9    int search_process(Char[] T, Char[] P, int i, int count)
10   begin
11           j'← P.length;
12           while (j'>=0 && T[ i + j'] == P[j'])
13           do
14                   j'←j'-1;
15           done ;
16           if (j'<0)
17                   i+= (i+m < n)? m- bad_char_array[T[i+m]] : 1;
18           else
19                   i+= max_process(1, j - bad_char-array[T[i+j']]);
20                   count++;
21           end if
22   return count;
23   end search_process;
/*max_process*/
24   public static int max_process(integer a, integer b)
25   begin
26   return (a > b)? a: b;
27   end max_process;
/* bad_char_rule*/
28   public static void bad_char_rule(char[] str, int size, int[] badchar)
```

```
29  begin
30          bad_char_array[i] = -1;
31          for i 1 to n
32          begin
33                  bad_char_array [str[i]] = i;
34          end
35  end bad_char_rule;
```

Algorithm 3: Skip Search String Matching Algorithm With Fork Join

Input: Monkey chromosome files, let these are considered as *Text files 'T' of length 'n' and Patterns (P_1, P_2, ...) of length m*

Output: *The number of occurrences and the positions of the patterns (P_1 , P_2, ...)*

```
/* Main */
1   n←T.length, m←P1.length or P2.length or ...or Pn.length
2   for each P1, P2,...Pn
3   for i ← 0 to n-m do
4   begin
5           fork_join(T);
6           count←search_process(T,P,i,count);
7   end for
8   end for
/* Search */
9   int search_process(Char[] T, Char[] P, int i, int count)
10  begin
11          j′← P.length;
12          j′← j′/2
13          j″← P.length;
14          while (j′>=0 && j″>=0 && T[i - j′] == P[j′])
15          do
16
17                  j′← j′/2 - 1;
18                  if (j′== -1)
```

```
19                    j'←(–) j''/2
20            done ;
21            if (j'== -1)
22                    count++;
23            end if
24    return count;
25    end search_process;
```

4. DATA SET

A monkey chromosomes contains 10 patterns called TAGA, TCAT, GAAT, AGAT, AGAA, GATA, TATC, CTTT, TCTG and TCTA, in this study these 10 patterns are considered as search patterns.

To assess the efficiency of the parallel brute force, boyer moore and skip search string matching algorithms, all the chromosomes of monkey (Cercocebus atys (771 mb)) dataset is considered[17]. The parallel brute force, boyer moore and skip search string matching algorithms are implemented in JAVA on WINDOWS 8.1 Operating System with 8GB of RAM. The experimental results of sequential brute force, boyer moore and skip search string matching algorithms are shown in Table 1. Parallel brute force, boyer moore and skip search string matching algorithms results are shown in Table 2. From these results (Table 1 and 2), graphs are drawn and shown in Figure 1 and 2.

Table 1
Search times of sequential brute force, boyer moore and skip search string matching algorithms

Patterns / *Mechanisms*	*TAGA*	*TCAT*	*GAAT*	*AGAT*	*AGAA*	*GATA*	*TATC*	*CTTT*	*TCTG*	*TCTA*	*Total*
Brute force	266219	255316	277489	211467	222888	276678	277890	299897	221654	263654	2573152
Boyer moore	256329	246426	267699	200577	213901	267000	268000	290130	212887	253780	2476729
Skip search	247302	238551	257611	193813	203012	256788	260201	279786	201655	247302	2386021

Table 2
Search times of parallel brute force, boyer moore and skip search string matching algorithms

Patterns / *Mechanisms*	*TAGA*	*TCAT*	*GAAT*	*AGAT*	*AGAA*	*GATA*	*TATC*	*CTTT*	*TCTG*	*TCTA*	*Total*
Brute force	66389	63670	69199	52735	55583	68997	69299	74787	55275	65749	641683
Boyer moore	64243	61761	67092	50270	53609	66917	67168	72714	53355	63604	620733
Skip search	61062	58901	63608	47855	50126	63404	64247	69083	49791	61062	589139

From the Figure 1, it is observed that, brute force, boyer moore and skip search string matching algorithms takes 2573152, 2476729, 2386021 milli seconds respectively to search 771 Mb of data for all ten patterns. From the Fig 1 it is also observe that Boyer moore string matching reduces search time than brute force, skip search algorithm reduces search time than Brute force and Boyer moore.

From the Figure 2, it is observed that, brute force, boyer moore and skip search string matching algorithms takes 641683, 620733, 589139 milli seconds respectively to search 771 Mb of data for all ten patterns. From the Fig 1 it is also observe that Boyer moore string matching reduces search time than brute force, skip search algorithm reduces search time than Brute force and Boyer moore as like sequential algorithms.

From the Figure 2, it is concluded that parallel brute force, boyer moore and skip search string matching algorithms performs very well compared to those of sequential string matching algorithms.

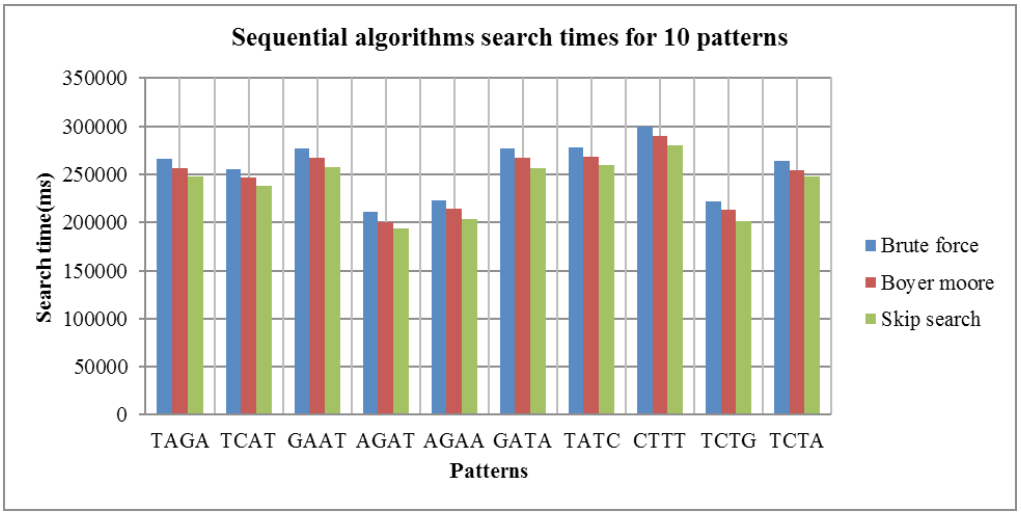

Figure 1: Search times of sequential Brute force, Boyer moore and Skip search string matching algorithms

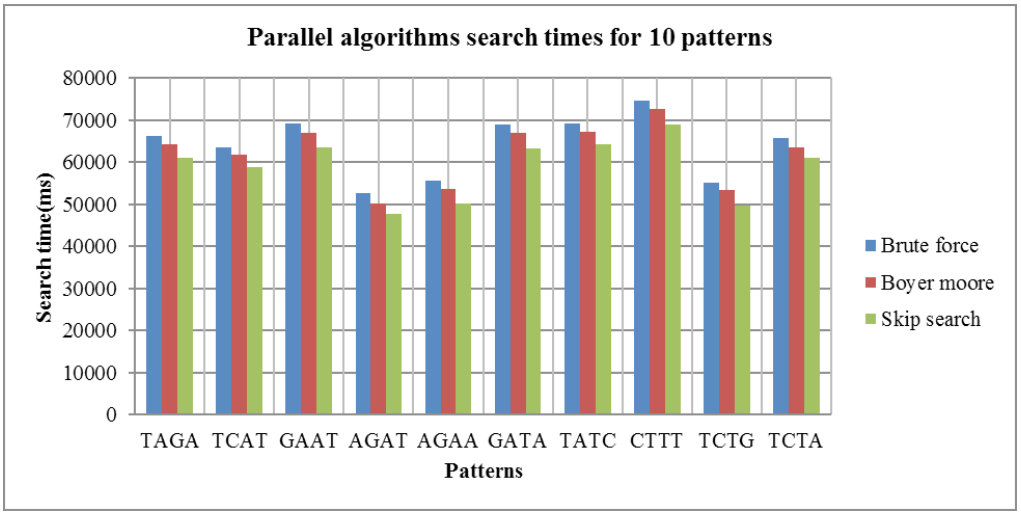

Figure 2: Search times of parallel Brute force, Boyer moore and Skip search string matching algorithms

5. CONCLUSIONS

In this paper, we have considered three retrieval string matching algorithms called Brute force, Boyer moore and Skip search, that are converted into parallel string matching algorithms. These parallel Brute force, Boyer moore and Skip search string matching algorithms are able to count each occurrence of the pattern in the entire file. To assess the efficiency of the parallel Brute force, Boyer moore and Skip search string matching algorithms , we have conducted experiments by taking Cercocebus atys(771 Mb) genome sequence as the data set and TAGA,

TCAT, GAAT, AGAT, AGAA, GATA, TATC, CTTT, TCTG and TCTA as patterns. From the experimental results observed that converted parallel string matching algorithms reduces search time as well when compared to those of original Brute force, Boyer moore and Skip search string matching algorithms.

REFERENCES

[1] Chinta Someswara Rao, Dr S Viswanadha Raju, "Next Generation Sequencing (NGS) Database for Tandem Repeats with Multiple Pattern 2^0-shaft Multicore String Matching", Genomics Data, Elsevier, Vol.7, 2016, PP.307–317, ISSN:2213-5960.

[2] Chinta Someswara rao, S Viswanadha Raju, **"**A Novel Multi Pattern String Matching Algorithm with While Shift", Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies, ACM, pp.1-5, 2016

[3] Chinta Someswara Rao, Dr S Viswanadha Raju, "Concurrent Information Retrieval System (IRS) for large volume of data with multiple pattern multiple (2^N) shaft parallel string matching", Annals of Data Science, Springer, Vol.3, Issue.2, 2016, PP.175-203, ISSN: 2198-5804.

[4] Chinta Someswara Rao, Dr S Viswanadha Raju, "A Frame Work for XML Ontology to STEP-PDM from Express Entities: A String Matching Approach", Annals of Data Science, Springer, Vol.3, Issue.4, 2016, PP.469-507, ISSN: 2198-5804.

[5] Chinta Someswara Rao, Dr S Viswanadha Raju, "Recent Advancements in Parallel Algorithms for string matching on computing models-a Survey and Experimental Results", ADCONS, Proceedings in LNCS Springer, 2012, pp. 270-278,ISBN: 978-3-642-29280-4.

[6] Chinta Someswara Rao, Dr S Viswanadha Raju, "Parallel String Matching with Multi Core Processors-A Comparative Study for Gene Sequences", Global Journal of Computer Science and Technology, Vol.13, Issue.1, 2013, PP.27-41, ISSN: 0975-4172.

[7] Aho, Alfred V., and John E. Hopcroft., "Design & Analysis of Computer Algorithms", Pearson Education India, 1974.

[8] R. S. Boyer and J. S. Moore, "A fast string searching algorithm", Communications of the ACM, vol. 20, no. 10, pp. 762-772, 1977.

[9] Charras, Christian, Thierry Lecrog, and Joseph Daniel Pehoushek. , "A very fast string matching algorithm for small alphabets and long patterns", InCombinatorial Pattern Matching, pp. 55-64. Springer Berlin Heidelberg, 1998.