Time Efficient String Matching Solution for Single and Multiple Pattern using Bit Parallelism

Vidya SaiKrishna
Department Of Computer Science And Engineering Maulana Azad National Institute Of Technology Bhopal-462051,

Akhtar Rasool
Department Of Computer Science And Engineering Maulana Azad National Institute Of Technology Bhopal-462051,

Nilay Khare
Department Of Computer Science And Engineering Maulana Azad National Institute Of Technology Bhopal-462051,

ABSTRACT

Bit Parallelism exploits bit level parallelism in hardware to perform operations. Bit Parallelism is a technique that is used to solve string matching problem, when the pattern to be searched for is less than or equal word size of a system. It is a technique that takes the advantage of intrinsic parallelism of the bit operations inside a system word. By using cleverly this fact, the number of operations that an algorithm performs can be cut down by a factor of at most w, the number of bits in system word. Since in current architecture word size is 32 bits or 64 bits, the speedup is very significant in practice. It is a form of parallel computing and is used to have a solution to exact string matching problem. The approach is further extended for multiple patterns string matching problem.

Keywords

String matching, Bit parallelism, Suffix Automata.

1. INTRODUCTION

In the recent years bit parallelism plays an important role in string matching , because 'w' length of the pattern can be processed in parallel. This is done by creating bit vectors of the pattern characters, and then the matching takes place with the help of bit operations in parallel. Transformation into bits results in more faster results as they can be performed in parallel. Bit parallelism although performs better as compared to other non bit parallel algorithms , but it imposes a limitation on the pattern size. Traditional algorithms solved using bit parallelism have a pattern size which is equal to the word length of the computer system. Therefore increasing the word size of the system , will make string matching algorithm work for patterns of larger size. Recent architecture makes use of 64 bit word size.

String Matching using bit parallelism can be viewed as being solved for single Pattern and multiple pattern. In single pattern string matching problem , there is a single pattern whose occurrence is to be reported in the text. In multiple pattern string matching problems, we are given a set of patterns whose occurrence's are to be reported in the text. The multiple pattern string matching problems are having more practical applications in real life.

The single pattern string matching problem using bit parallelism technique is a simulation of a method that constructs suffix automaton. This method has certain advantages. Firstly it eliminates the need of constructing automaton thereby reducing the memory requirement and secondly the time requirement is also reduced because of the machine inherent property of bit parallelism.[1] The method is called Bit Parallel Automaton and it uses the algorithm called BNDM(Backward Non Deterministic Matching) which is compared with Boyer Moore string matching algorithm and KMP(Knuth Morris Pratt) algorithm whose experimental results are shown.

The BNDM algorithm has numerous variations to further reduce the time requirement. One of the variations called TNDM(Two-Way Non deterministic matching) performs two way searching , which results in longer shifts in the text.[5] Another variation described is a simplified BNDM algorithm which results in reduced running time. The BNDM algorithm is also modified to work for longer patterns. The variations are briefly described.

2. BIT PARALLELISM

Bit Parallelism is an intrinsic feature of system, which uses bit array or bit vector. A bit vector is a data structure [9] which stores individual bits in a compact form and is effective at exploiting bit level parallelism in hardware to perform operations quickly as well as reducing memory requirements. Bit Parallelism can be viewed as kind of parallel computing , which makes use of the word size of the computer. This is because, the processor is capable of processing the entire word in one memory cycle. To understand this , consider a simple example. A 16 bit processor will have to perform two memory cycles to perform addition of two 32 bit numbers. In the first cycle , the lower order 16 bit numbers are added and then the second memory cycle performs addition of next higher order 16 bit numbers.

If string matching problem can be solved making use of the bit parallelism technique, it will result in faster matches. The method imposes a limitation on the pattern size. The pattern size is limited to the word length of the system. The following section describes the method to solve the string matching problem using bit parallelism. The method uses an algorithm called BNDM(Backward Non Deterministic Matching) which makes use of bit parallel automaton.

2.1 Bit Parallelism in Single Pattern Matching

BNDM (Backward Non Deterministic Matching) is the simulation of BDM(Backward Deterministic Matching) algorithm which uses the concept of Suffix Automaton. After a brief description to the method using suffix automaton , the bit parallel approach is described.

Concept of Suffix Automaton

A *suffix automaton* on a pattern $P_{1\ldots m}$ is a deterministic finite automaton that recognizes the suffixes of *P*.[1]The Suffix Automata can be built in O(m) time. To search a text *T* for *P* , the suffix automaton of $P^r = PmPm-1 \ldots P1$ (the pattern read backwards) is built.

An **Example** to know how the method using suffix automaton works is explained below in following steps:
Pattern : **"k o o b"**
Text : **"o k b o k o o b o o"**
Step 1 : Construct the NFA with ϵ moves corresponding to p^r as shown in figure 1.

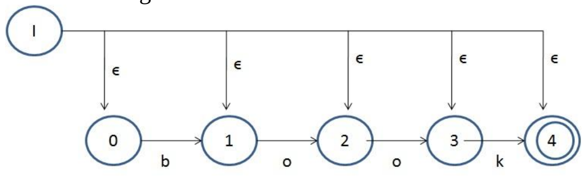

Figure 1: NFA corresponding to p^r= "book"

Step 2 : The NFA is converted into DFA as shown in figure 2, which is known as DAWG(Directed Acyclic Word Graph)

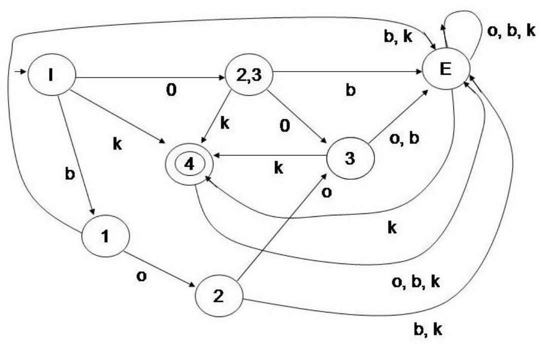

Figure 2 : DAWG for P^r= "book"

As clear from the above automaton , it recognizes all the suffix's of the reverse of the pattern. The suffixes are "k", "ok", "ook" and finally "book" which is accepted in the automaton.

BDM Algorithm

The BDM (Backward Deterministic Matching) uses the suffix automaton to perform string matching in the text.
A window of length *m* is slid along the text, from left to right. The algorithm reads the window right to left and feeds the suffix automaton with the characters read. During this process, if a final state is reached, this means that the window suffix traversed is a prefix of *P* (because suffixes of P^r are reversed prefixes of *P*). Then value of the current window position is stored in the variable *last*, possibly overwriting its previous value. The backward window traversal ends in two possible forms:[2]

(1) A factor is not recognized i.e a character σ is reached that does not have a transition in the automaton. In this case the window suffix read is not a factor of *P* and therefore it cannot be contained in any occurrence. The window is shifted to the right, aligning its starting position to *last*, which corresponds to the longest prefix of *P* seen in the window. An occurrence cannot be missed because in that case the suffix automaton would have found its prefix in the window.[2]

(2) Beginning of the window is reached , thereby recognizing the pattern *P* . An occurrence is reported and the window is shifted exactly in the previous case. [2]

Example: Matching the pattern with the text as shown in table 1.
T= [o k b o] k o o b o o , m=4, last=4

Table 1: Suffix automata search example

1.	T=[o k b o] k o o b o o , o is Factor of P^r and final state is not reached , therefore last remains as it is i.e Last=4	4.	T= o k b o[k o o b] o o , "o b" is a factor of P^r and final state is not reached , therefore last remains as it is i.e last= 4
2.	T= [o k b o] k o o b o o , fails to recognize next b. Therefore backward traversal ends and the window is shifted by last position.	5.	T= o k b o[k o o b] o o "o o b" is a factor of P^r and final state is not reached , therefore last remains as it is i.e last= 4
3.	T=o k b o[k o o b] o o, "b" is a factor of P^r and final state is not reached , therefore last remains as it is i.e last= 4	6.	T=o k b o[k o o b] o o "ko ob" is a factor of P^r and we reach final state , also the beginning of the window . We recognize the word "koob" and report an occurrence.

A bit parallel version of the algorithm is used which eliminates the need of constructing the automata and performs in the same manner . The algorithm is called BNDM (Backward Non Deterministic Matching).

Simulation of BDM algorithm using BNDM algorithm

The basic idea of BNDM is that it maintains a set of position on the reverse pattern that are beginning positions of the factors of the text positioned in the window. The set is stored as 0 and 1 where 1 denotes occurrence. The vector D keeps a list of positions in pattern where the factor begins. This is shown in figure 3.

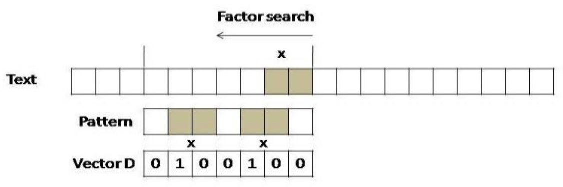

Figure 3: Bit parallel Factor search

Each time the window is positioned in the text, D is initialized and the window is scanned backwards. For each new text character update D. $D=d_m d_{m-1} \ldots\ldots d_1$ which keeps the state of search using m bits of Computer word. Whenever a prefix is found in the pattern (d_m=1), its position is remembered. If the value in D becomes zero, then there cannot be a match and we suspend scanning. If m iterations are performed then a match is reported. The bit mask is stored in bit vector B. This mask sets the corresponding to positions 1 where p_i=c.[5]
The formula to update D is : $D' \leftarrow (D \ \& \ B[t_j]) << 1$
The simulation is done in the following manner.

- One in MSB position of Vector D corresponds to the condition that a factor has been identified and there is a transition to final state of DAWG.
- Zero in MSB position of vector D corresponds to the condition that there is transition to non final state .
- All zero values in D corresponds to the condition that there is no transition from the present state to other state. In this case the window is shifted by the value stored in last position.

BNDM Algorithm

BNDM($p=p_1p_2\dots pm$, $T=t_1t_2\dots t_n$)[3]

```
1.  Preprocessing
2.  For $c\in\sum$ do $B[c] \leftarrow 0^m$
3.  For $i\in 1\dots m$ do $B[p_{m-i+1}] \leftarrow B[p_{m-i+1}] | 0^{m-i}10^{i-1}$
4.  Search
5.  pos$\leftarrow$0
6.  while pos<=n-m do
7.      j$\leftarrow$m, last$\leftarrow$m
8.      $D=i^m$
9.      While $D!=0^m$ do
10.         D$\leftarrow$D & $B[t_{pos+j}]$
11.         j$\leftarrow$j-1
12.        If D & $10^{m-1}$ $!=0^m$ then
13.         If j>0 then last$\leftarrow$j
14.            Else report an occurrence at pos +1
15.       End of if
16.       D$\leftarrow$D<<1
17.     End of While
18.     Pos=pos+last
19. End of while
```

An **Example** shows how the algorithm works and the working is shown in table 2:

T=[o k b o] k o o b o o D=1111

B[b]=0001 B[o]=0110 B[k]=1000, m=4, last=4, j=4. Pattern: “ koob”

Table 2: BNDM example

1.	T=[o k b o] k o o b o o 1 1 1 1 j=3 & 0 1 1 0 D=0 1 1 0	5	T= o k b o[k o o b] o o 0 1 0 0 j=1 & 0 1 1 0 D=0 1 0 0
2.	T=[o k b o] k o o b o o 1 1 0 0 j=2 & 0 0 0 1 last=4 D=0 0 0 0 We fail to recognize next b so shift window by last position	6	T=o k b o[k o o b] o o 1 0 0 0 & 1 0 0 0 last =4 D=1 0 0 0 j=0 so occurrence is reported at pos 5.
3.	T=o k b o[k o o b] o o 1 1 1 1 j=3 & 0 0 0 1 D=0 0 0 1	7.	Shift the window by pos+last position , which is equal to 4+4=8. The main while loop terminates as pos>(10-4)
4.	T= o k b o[k o o b] o o 0 0 1 0 j=2 & 0 1 1 0 D=0 0 1 0		

Analysis of BNDM Algorithm

- Worst case time complexity of the Backward Nondeterministic DAWG Matching (BNDM) is O(nm). This is because in the worst case the window will be shifted by one character position , and also in a fixed window mismatch occurs when the last character is scanned .
- Handle class, multiple pattern, and allow errors
- Using bit parallelism, Combine Shift-AND and BDM
- Faster than BDM , Faster than BM [3]
- Update Function D’$\leftarrow$ (D & $B[t_j]$) << 1

Variation in BNDM Algorithm

Several modifications have been done in BNDM algorithm that extends its capacity to search the pattern in the text in lesser time and also to search for pattern that is longer than the word size of the computer.

One of the algorithms is a two way modification of BNDM also called Two-way Non Deterministic DAWG Matching(TNDM). In this method if the text character aligned with the end of the pattern is a mismatch , BNDM scans backwards in the text if the conflicting character occurs elsewhere in the pattern . In such situation TNDM will scan forward , i.e it continues by examing text characters after the alignment.[5]The change of direction will decrease the number of examined characters.The forward scan ends until a suffix of the pattern has been found.When the suffix is found in the forward direction , the window is shifted to text position of the found suffix. Working in this manner will decrease the number of examined characters, and will reduce the time for searching.This is illustrated in figure 4.

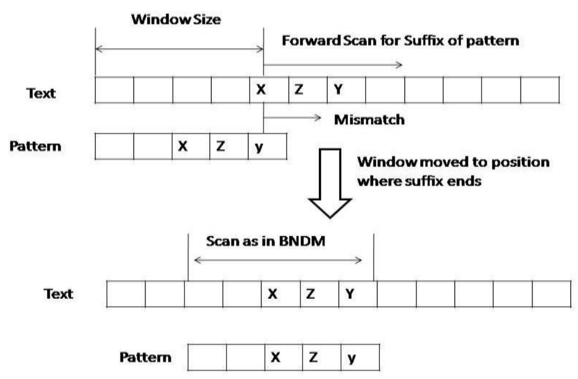
Window Size
Forward Scan for Suffix of pattern
Text
X Z Y
Mismatch
Pattern
X Z y
Window moved to position where suffix ends
Scan as in BNDM
Text
X Z Y
Pattern
X Z y

Figure 4 : TNDM Pattern Search

ALGORITHM TNDM($p=p_1p_2\dots p_m$, $T=t_1t_2\dots t_n$)

```
1.  For $c\in\sum$ do $B[c] \leftarrow 0^m$
2.  For $i\in 1\dots m$ do $B[p_{m-i+1}] \leftarrow B[p_{m-i+1}] | 0^{m-i}10^{i-1}$
3.  Init_shift(P,restore[ ])
4.  epos$\leftarrow$m
5.  while epos<=n do
6.        i$\leftarrow$0;last$\leftarrow$m
7.        D$\leftarrow$$B[t_{epos}]$
8.       if (D & 1)=0 then
9.           do /*forward scan for the suffix */
10.             i$\leftarrow$i+1
11.             D$\leftarrow$D & ($B[t_{epos+i}]$ ]<<i)
12.             While ( D $\neq$ 0 and D & $10^i=0^m$)
13.              if $D=0^m$ then
14.              Goto over
15.              epos$\leftarrow$epos + i; last$\leftarrow$restore[i]
16.      do
17.        i$\leftarrow$i+1
18.        if D & $10^{m-1}\neq 0^m$ then
19.           if i<m then last$\leftarrow$m-i
20.           else report an occurrence at epos–m
              +1 ;goto over
21.         D$\leftarrow$D<<1
22.         D$\leftarrow$D & $B[t_{epos-i}]$
23.       while D $\neq$ $o^m$
24.       over: epos$\leftarrow$epos+last
25.       end of while
```

ALGORITHM Init_shift ($p=p_1p_2\dots p_m$, restore[ ])

```
1.   D ←1m
2.   last←m
3.   for i=m downto 1
4.        D=D & B[pi]
5.        if D & 10m-1 ≠ 0 then
6.             if i>0 then last←i
7.        restore[m-i+1] ←last
8.        D=D<<1
```

Example: Text : “obookookbook”
Pattern : “book”
Suffix of the Pattern are “k”, “ok”, “ook”, “book”
The underlined character denotes the last character fetched. The working is explained in table 3.

Table 3 :TNDM Search Example

Text Window	D	i	epos	Last	Explanation
oboo	0110	0	4	4	Lowest bit is 0, so forward scan for suffix.
obook	0010	1	4	4	D ≠ 0 and D & 10 ≠ 0, so leaves the loop.
book	0010	1	5	4	Suffix found, epos and last are updated.
book	0100	2	5	2	D ≠ 0
book	1000	3	5	1	A match reported, epos updated.
ooko	0110	0	6	4	Lowest bit is 0, so forward scan for suffix.
ooko	0100	1	6	4	D ≠ 0, D & 10 = 0
ookook	0100	2	6	4	D ≠ 0, D & 100 ≠ 0, leaves the loop, epos and last are updated.
ookook	1000	3	8	4	D & 1000 ≠ 0.
ookook	0000	4	8	4	D = 0, leaves the loop, epos updated.
book	0001	1	12	4	Lowest bit is 1.
book	0010	2	12	4	D & 1000 = 0
book	0100	3	12	4	D & 1000 = 0.
book	1000	4	12	4	D & 1000 ≠ 0, i = m, occurence reported.

Experimental results prove that TNDM reduces the number of examined characters and thus results in faster searching.

3. BIT PARALLELISM IN MULTIPLE PATTERN MATCHING

The Bit parallel approach can be extended to search for multiple patterns inside the text. The method also works for larger pattern sets. For large pattern sets , the bit parallel approach can be beneficial in terms of execution speed and memory requirement. In reference [8] , bit parallel approach for multipattern sets is described.

The method uses a bit vector B[c] which is initialised in a way such that the ith bit is 0 if the character appears in any of the patterns in position i . The automaton has a transition from state i to state i + 1 on character c if ith bit in B[c] is 0. Another vector D is used which is initialized to all 1’s. When the character c is read from the text D is updated as D = (D<<1) | B[c] . After the update, ith bit in D is 0 if i − 1th bit was 0 (the previous state i − 1 was active) and ith bit is 0 in B[c] (there is a transition from state i − 1 to i on c).

The assumption in this method is that all the patterns $p_1p_2…..p_r$ have equal size m and m<=w, where w is word size of the computer.[8]

ALGORITHM MultiplePatternSearch(text=t1t2..tn)

```
1.   POS←0.
2.   textLength←stringLength(text)
3.   N←0// N denotes Number Of Occurrence
4.   D=1^m
5.   While(POS<= textLength)
6.        D=D<<1 | text[B[POS]]
7.        If(MSB[D]=0)
8.             POS←POS+1
9.             N←N+1
10.            GOTO Step 4
11.       POS←POS+1
12.  End of While
```

Example : Text= “hhello”
Pattern = { “hello”, “world”}
The Bit Vectors are set in the following manner.

B[h]=11110, B[e]=11101,B[l]=10011,B[o]=01101 , B[w]=11110, B[r]=11011, B[d]=01111
The Automaton recognizing the set of patterns is shown in figure 4

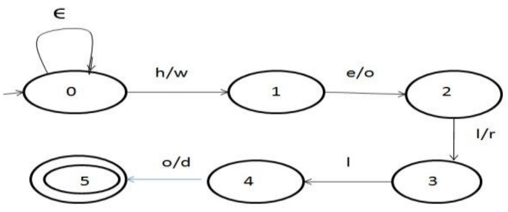

Figure 4 : NFA finding occurrence of character class pattern.

The character class pattern is “[h,w],[e,o],[l/r],[l],[o/d]”
Table 2 shows bit parallel simulation of above automata.

Table 2 : Multiple pattern search example

1	Text = hhello D 11110 B[h] 11110 OR D 11110 D[0]=0 , so shift To next state	3.	Text = hhello D 11100 B[e] 11101 OR D 11101 D[1]=0, so shift to next State
2.	Text = hhello D 11100 B[h] 11110 OR D 11110 D[1]=1 , so it remains in the same state	4.	Text = hhello D 11010 B[l] 10011 OR D 11011 D[2]=0, so shift to next state
5.	Text = hhello D 10110 B[l] 10011 OR D 10110 D[3]=0, so shift to next state	6.	Text = hhello D 01100 B[o] 01101 OR D 01101 D[4]=0, so shift to next State, which is the final state And the pattern is recognized.

The method used for multiple pattern search is based on filtering approach. The filter method works in three phases. In the first phase , the pattern is preprocessed. In the second phase, matching takes place and in the third phase the matches generated by the method needs to be verified for more accurate results.

The filtering process can be improved by using Q-grams of the pattern.[8]. In this method ,the pattern is first transformed into a sequence of Q-grams. After this filtering takes place with the character class patterns built from the transformed pattern set. The matches are finally verified using Rabin Karp method.[8]

4. EXPERIMENTAL RESULTS

Experimental results have proven that Bit Parallel Automaton using the BNDM algorithm results in more better results than the Boyer Moore Algorithm . The Experiment was conducted on patterns of different sizes and the time is reported in seconds .The following experimental conditions have been used.

Experimental Conditions:

Processor : Intel Core i7-260 M CPU,2.80 Ghz
RAM : 8 GB
System Type: 64 Bit Operating System
OS : Windows 7 Professional
Text Size : 230 MB
Pattern Size : 4,6,8,10,12 characters

The following graph shown in figure 4 shows a comparative Analysis of Boyer Moore algorithm and BNDM algorithm along with tabular result.

Table 4 : Performance Comparison of Boyer Moore and BNDM

Pattern Size In Characters	Boyer Moore	BNDM
	Time In Milliseconds	
4	374	343
6	203	168
8	328	272
10	368	296
12	141	94

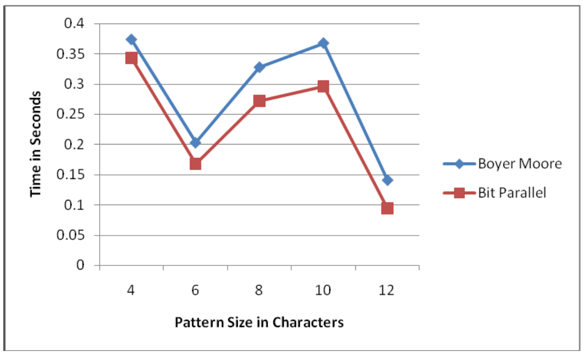

Figure 5 : Comparative Analysis of Boyer Moore and BNDM.

The TNDM algorithm is also compared with the BNDM algorithm and the experiment was conducted on a text size of 265 MB. In the experiment we have considered the text in which there are very small occurrences of the pattern . In such case , the TNDM algorithm will reduce the number of examined characters and thus results in faster searching. Figure 6 shows the comparative analysis of both the algorithms.

Table 5 : Performance Comparison of TNDM and BNDM

Pattern Size in Characters	TNDM	BNDM
	Time in Milliseconds	
4	125	178
6	103	125
8	78	135
10	93	109
12	94	102

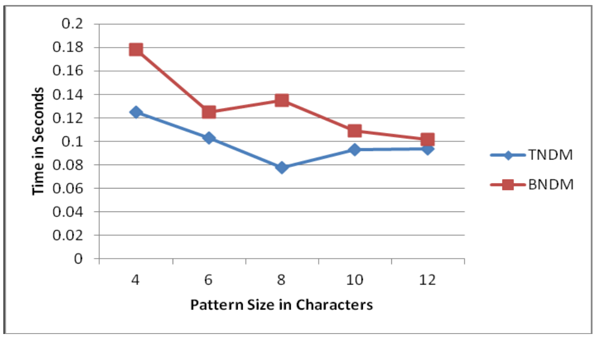

Figure 6 : Comparative Analysis of TNDM and BNDM.

The bit parallel multiple pattern string matching algorithm is also compared with the traditional Aho-Corasick algorithm. Both the algorithms give very small variations in searching time , when worked for different number of patterns, but the speed up achieved using bit parallel approach is about 1.0 as compared to Aho-Corasick algorithm. Figure 7 shows the comparative analysis of both the algorithms, when executed on a text size of 273 MB.

Table 6 : Performance Comparison of Aho Corasick and BNDM

Pattern Size In Characters	Aho Corasick	BNDM
	Time In Milliseconds	
3	1049	426
6	1050	399
9	1057	410
12	1060	450

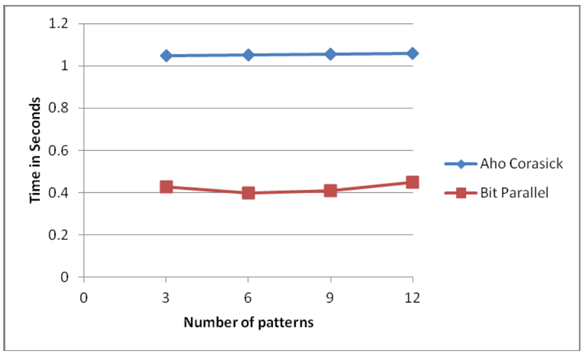

Figure 7 : Comparative Analysis of Aho Corasick and Bit parallel Multiple pattern algorithm.

5. APPLICATION AREAS OF BIT PARALLEL STRING MATCHING

In general bit parallel string matching algorithms are the most efficient as compared to the other algorithms. The main idea of the bit-parallel algorithms is that they store several data items into a single computer word and then update them in parallel using a single computer operation . String matching is often used in different areas such as text editors, virus scanning, digital libraries , web search engines, intrusion detection.

Bit-parallel algorithms are very efficient for approximate string matching. This problem has many applications in computational biology viz. finding DNA subsequences after possible mutations, locating positions of a disease(s) in a genome etc.

6. CONCLUSION

Experimental results have proven that the Bit Parallelism approach to string matching increases the speed of matching as compared to Suffix automata which uses the BDM algorithm. It is also faster as compared to Boyer-Moore algorithm.

Several variations of BNDM algorithm further improve the performance of the traditional BNDM algorithm. The bit parallel approach for solving the multiple pattern string matching ,also performs better than the Aho-Corasick algorithm, whose experimental results are shown. The performance of Aho-corasick algorithm degrades as the pattern size increases , as depending upon the pattern size , the size of the trie grows enormously. The bit parallel algorithm becomes practical in applications where we have a large pattern set ,such as application including intrusion detection, bioinformatics and antivirus scanning.

7. FUTURE WORK

In future the BNDM algorithm and the multiple pattern bit parallel algorithm can be implemented on a GPGPU (Graphics Processing Unit) for even faster performance.

8. REFERENCES

[1] Gonzalo Navarro and Mathieu Raffinot. A Bit Parallel approach to Suffix Automata : Fast Extended String Matching. In M. Farach (editor), Proc. CPM'98, LNCS 1448. Pages 14-33, 1998.

[2] G. Navarro,M. Raffinot, Fast and flexible string matching by combining bit-parallelism and suffix automata,ACM J. Experimental Algorithmics (JEA) 5 (4) (2000).

[3] M. Crochemore et al., A bit-parallel suffix automaton approach for (δ, γ)-matching in music retrieval, in: Proc. 10th Internat. Symp. on String Processing and Information Retrieval (SPIRE'03), in: Lecture Notes in Computer. Sci., vol. 2857, 2003, pp. 211–223

[4] R. Baeza-Yates, G. Gonnet, A new approach to text searching, Comm. ACM 35 (10) (1992) 74–82.

[5] Hannu Peltola and Jorma Tarhio , Alternative Algorithms for Bit-Parallel String Matching, String Processing and Information Retrieval, 2003 - Springer

[6] http://en.wikipedia.org/wiki/bit-level_parallelism

[7] http://en.wikipedia.org/wiki/string_searching_algorith m

[8] Leena Salmela, J. Tarhio and J. Kytojoki "MultiPattern String Matching with Very Large Pattern Sets", ACM Journal of Experimental Algorithmics, Volume 11, 2006.

[9] G. Myers , " A fast bit-vector algorithm for approximate string matching based on dynamic programming ", J. ACM, 46 (3) (1999), pp. 395–415

[10] G. Novarro, " A guided tour to approximate string matching", ACM Comput. Surv., 33(1)(2001),pp 31-88

[11] Heikki Hyyro, Kimmo Fredrikson, Gonzalo Novarro, "Increased Bit Parallelism for Approximate and Multiple String Matching", Journal of Experimental Algorithmics, Vol 10 , 2005