Approximate Multiple Pattern String Matching using Bit Parallelism: A Review

Syed.Danish.Ali
Department of Computer Science & Engineering
All Saints' College of Technology

Zuber Farooqui
Department of Computer Science & Engineering
All Saints' College of Technology

ABSTRACT

String matching is to find all the occurrences of a given pattern in a large text both being sequence of characters drawn from finite alphabet set. Approximate String Matching involves the detection of correct patterns along with the detection of some wrong patterns inside the text. Bit Parallelism is a feature that can be used to detect patterns inside the text and is reported to result in more efficient approximate string matching. Bit parallelism enhances the processing speed of the approximate string matching algorithm as it takes the benefit of the internal bit operations taking place in parallel inside the system. The bit parallel method has also been compared with the traditional Aho Corasick Algorithms which consumes more time and memory. In general bit parallel are both memory and time efficient.

Keywords

Approximate String matching, Bit parallelism, Shift OR String Matching.

1. INTRODUCTION

String matching consists in Finding one or more generally all the occurrences (exact or approximate) of a single string (or a finite set of strings) in a text. It is an extensively studied problem in computer science, mainly due to its direct applications to such diverse areas as text, image and signal processing, speech analysis and recognition, information retrieval, computational biology and chemistry[8]. String matching is a very important subject in the wider domain of text processing and algorithms for the problem are also basic components used in implementations of practical software's existing under most operating systems. Moreover, emphasize programming methods that serve as paradigms in other fields of computer science. Finally also play an important role in theoretical computer science by providing challenging problems.

In the recent years bit parallelism plays an important role in string matching, because 'w' length of the pattern can be processed in parallel [8]. 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 faster results as 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 has 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 [8].

String Matching using bit parallelism can be viewed as being solved for single Pattern and multiple patterns. 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, there 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.

2. BIT PARALLELISM

Bit-parallelism is a technique which takes advantage of the intrinsic parallelism of the bit operations inside a computer word, allowing to cut down the number of operations that an algorithm performs by a factor up to the number of bits in the computer word[8]. Bit-parallelism is indeed particularly suitable for the efficient simulation of non-deterministic automata. In other words, Bit-parallelism is the technique of packing several values in a single computer word and updating them all in a single operation. This technique has yielded the fastest approximate string-matching algorithms if exclude filteration algorithms [8].

Bit-parallelism permits executing several operations simultaneously over a set of bits or numbers stored in a single computer word. This technique permits for the approximate searching of a pattern of length n in O(m/wn), where w is the number of bits in the computer word. Using bit parallelism multiple patterns can be packed into a single computer word so as to search for all them simultaneously. Instead of spending O(rn) time to search for r patterns of length $m \leq w/2$, require O(rm/wn) time. Second, the mechanism permits boosting the search for a single pattern of length $m \leq w/2$. Finally, the ideas can be applied to other problems such as single and multiple pattern string matching.

3. SINGLE PATTERN MATCHING USING BIT PARALLELISM

BNDM (Backward Non deterministic Matching) is Bit Parallel Simulation of the BDM (Backward Deterministic Matching) that uses the concept of Suffix Automaton [3, 4]. 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 1.

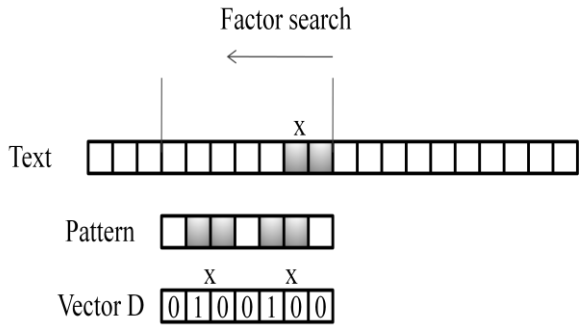

Figure 1: Bit parallel Factor search

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=dmdm-1……d1 which keeps the state of search using m bits of Computer word. Whenever a prefix is found in the pattern (dm=1), its position is remembered. If the value in D becomes zero, then there cannot be a match and scanning is suspended. 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 pi=c.[3]

The formula to update D is : D'← (D & B[tj]) << 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 ($P=p_1p_2p_3…p_m$, $T=t_1t_2t_3…t_n$) [3]

1. Initialization

s=1,j=0,d=1^m;

2. Pre-processing

For i=0….m-1 do B[Pm]=0
For i=0….m-1 do B[Pm]=B[Pm]|s;
s = s<<1;

3. While j<=n-m do
i=m-1, last=m,d=1^m;
while d!=0m and i>=0 do
d = d &b[txt[j+i]] i = i-1
if d!=0^m do
if i>=0 do
last=i+1;
else do
count=count+1;
report occurrence of pattern
end else
end if
d=d<<1;
end while
j=j+last
end while.

An **Example** shows how the algorithm works and the working is shown in table 1:

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"

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 It 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		

Table 1: BNDM example

3.1 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'← (D & B[t_j]) << 1

4. MULTIPLE PATTERN MATCHING USING BIT PARALLELISM

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. The bit parallel approach for multipattern sets uses the Shift OR Algorithm for locating the patterns inside the text.[8]

The method uses a bit vector B[c] which is initialised in a way such that the i^{th} 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, i^{th} bit in D is 0 if i − 1^{th} 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)[8].

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 ShiftOr(text=t1…tn, patterns=p1,..,pk)[8]

1. **Initialization**
   m= pattern length, s=1, count=0,
   position=0,
2. **Preprocessing**
   [text[i]] <- 1^m
   for j= 0…k do
   for i= 0… m-1 do B[ptn[j][i]] <-
   B[ptn[j][i] & ~ (s<<1)
   end for
   end for
3. While pos< n do
   D = D <<1 & 1^m
   D=D | B[text[pos]]
   if D> 1^{m-1} do pos<- pos + 1
   Else do count <- count +1
   Report occurrence at position pos<- pos-m +1
   D <- 1^m
   pos<- pos +1
   end else

end 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 2

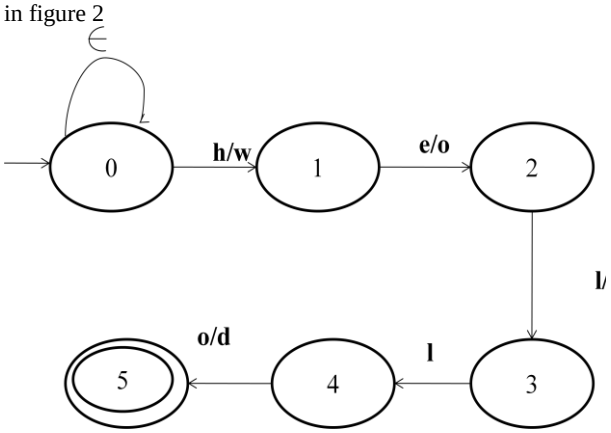

Figure 2 : NFA finding occurrence of character class pattern.

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

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.

Table 2 : Multiple pattern search example

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 [8].

4.1 ANALYSIS OF Shift OR ALGORITHM

- If the Text Length is assumed to be n , then the patterns are processed in O(n) time complexity.
- All the patterns are assumed to be of uniform length and less than or equal to the word size of the system.
- The method is a filter where the potential matches needs to be verified.
- Number of False Matches for Shift OR Method

It is assumed that there is a pattern set P=(p1,p2……pk) of K patterns. All the patterns are assumed to be having equal length m. The false matches are calculated for the worst case , where all the patterns are assumed to be having distinct characters in all pattern positions[12] . In this case:

(i) Total Number of correct Matches (CM) = K, as recognized by the Automaton.

(ii) Total number of matches recognized by the automaton (TM)= K^m

(iii) Total Number of false matches (FM1) = Total Matches- Total number of Correct matches.
FM= K^m – K

(iv) In addition to these there are other false matches detected . Considering the following text and the pattern
Text: "heabcdello" and the pattern "hello".
The Shift OR method will detect one pattern match in the above text. Counting the false matches for such case.
FM2= m(∑*-k) where ∑ denotes the size of the input alphabet .

(v) Total False Matches(FM)= FM1 + FM2
FM= K^m – K + m(∑*-k)
= K^m – K + m∑* - mk
= K{ K^{m-1} –m-1} + m∑*

5. EXPERIMENTAL RESULTS

An Experiment was performed to compare the performance of Multiple Pattern Shift OR Algorithm that uses approximate matching and the Aho Corasick Algorithm. The Comparison was done on the basis of number of matches detected by both the algorithms and the time factor separately. The Aho Corasick algorithm finds the matches correctly but the time taken is 10-20% more than the time taken by the Multiple Pattern Shift OR Algorithm. Also the as the pattern set size grows , the Aho Corasick algorithm uses enormous memory to build the Trie. The number of matches detected in Multiple Pattern Shift OR Algorithm is 20-30% more than the Aho- Corasick Algorithm. The following experimental conditions were used for the experiment.

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 : 350 MB
Pattern Size : 20, 40 , 60 , 80 and 100 characters

The Comparison between the algorithms is done in both tabular and graphical manner. Table 3 shows the tabular comparison between Shift OR and Aho Corasick algorithm based on the number of Matches and table 4 shows the comparison done on the basis of time factor reported in milliseconds.

Pattern Size in Characters	Aho Corasick	Shift OR
	Number of Matches	
20	2569988	3263885
40	3556765	4517092
60	4567765	5801062
80	6776523	8606184
100	7654355	9721031

Table 3 : Number of Matches Detected

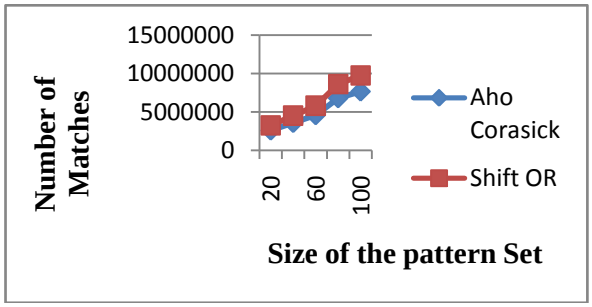

Figure 3 : Comparative Analysis of Aho Corasick and Bit parallel Multiple pattern algorithm.

Table 4 : Time Analysis of both Aho Corasick and Shift OR

Pattern Size in Characters	Shift OR	Aho Corasick
	Time in Millisec	
20	180	207
40	197	227
60	230	265
80	367	422
100	463	532

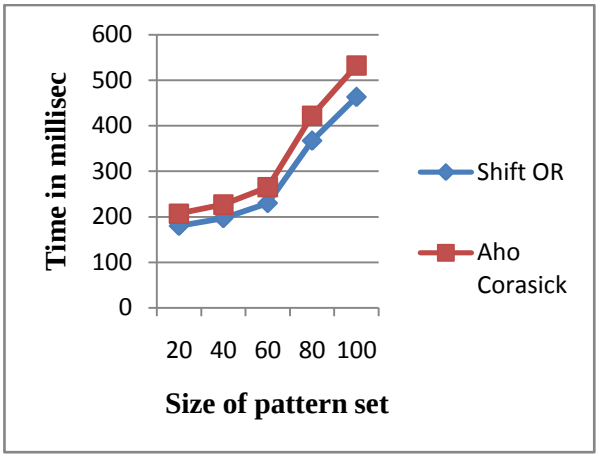

Figure 4 : Time Analysis of Aho Corasick and Bit parallel Multiple pattern algorithm.

6. 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 to store several data items into a single computer word and then update them in parallel using a single computer operation [5]. 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[8].

7. PROPOSED WORK

The Bit Parallel approach for locating the patterns inside the text is a filter where the potential matches needs to be verified. In future will try to improve the filtering efficiency of the shift OR method by considering n grams of the pattern characters. In this direction will first try to construct 2 grams of the pattern set. Constructing 2 grams will require an 2 dimensional array of size 256*256 to be constructed. The size of the automaton will also reduce which in other in other sense would mean that the text will be processed faster. This is because now 2 characters will be read at a time as compared to single character in previous method. The filtering efficiency would also improve the design would generate less false combinations. Further improvement can be done if 3 grams of the pattern are considered. The filtering efficiency and the speed improve at the cost of increasing the memory requirement because to consider n grams it requires an n dimensional memory to be constructed.

8. CONCLUSION

The bit parallel approach for solving the multiple pattern string matching 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 tries grows enormously. The bit parallel algorithm becomes practical in applications when there is a large pattern set, such as application including intrusion detection, bioinformatics and antivirus scanning.

9. ACKNOWLEDGMENT

I would like to thanks to my guide Prof Zuber Farooqui who has give his precious time and guideline in order to complete this paper. This paper will never complete without the help of Department of Computer Science & Engineering, ASCT Bhopal.

10. REFERENCES

[1] Rajesh Prasad, Suneeta Agarwal, Ishadutta Yadav, Bharat Singh "Efficient Bit-Parallel Multi-Patterns String Matching Algorithms for Limited Expression", Compute 2010 ACM

[2] Heikki Hyyr¨o, Kimmo Fredriksson Gonzalo Navarro "Increased Bit-Parallelism for Approximate and Multiple String Matching", ACM Journal of Experimental Algorithmics Vol 10 2006.

[3] 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.

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

[5] 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

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

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

[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] AHO, A. AND CORASICK, M. 1975. Efficient string matching: n aid to bibliographic search. Communications of the ACM 18, 6, 333–340.

[10] HYYR¨O, H. AND NAVARRO, G. 2002. Faster bit-parallel approximate string matching. In Proc. 13th Combinatorial Pattern Matching (CPM '02). LNCS 2373. Berlin, Germany, Springer, New York.203–224.

[11] NAVARRO, G. AND RAFFINOT, M. 2000. Fast and flexible string matching by combining bitparallelism and suffix automata. ACM Journal of Experimental Algorithmics (JEA). 5, 4.

[12] Vidya Saikrishna, Akhtar Rasool and Nilay Khare. Article: Spam Filtering through Multiple Pattern Bit Parallel String Matching Combining Shift AND & OR. International Journal of Computer Applications 61(5):40-45, January 2013. Published by Foundation of Computer Science, New York, USA.