Efficient String Matching Using Bit Parallelism

Kapil Kumar Soni, Rohit Vyas, Dr. Vivek Sharma

TIT College, Bhopal, Madhya Pradesh, India

Abstract: Bit parallelism is an inherent property of computer to perform bitwise parallel operation on computer word, but it is performed only on data available in single computer word. Bit parallelism inherently favors parallelism of bit operations within computer word. Parallel computing comprises bit parallelism and analyzed that it can be carried out “in parallel” which ensures utilizing the word size of computer. This technique is being properly utilized to work out on various string matching problems for increasing the efficiency of various real world applications. Since 1992 bit parallelism is being used in string matching applications to improve the matching pace. There are a variety of important bit parallel string matching algorithms exist like Shift-OR, BNDM, TNDM, SBNDM, BNDMq, Shift-OR with Q-Gram, and Multiple Patterns BNDM. This paper discusses the various important bit parallel string matching algorithms by means of example along with their advantages and disadvantages.

Keywords: String Matching, Bit Parallelism, Shift-OR, BNDM, TNDM, SBNDM, BNDMq, Shift-OR with Q-Gram, Multiple pattern BNDM.

I. INTRODUCTION

Bit parallelism [1] is a built in quality of computer in which basic bit wise operations like AND, OR, NOR etc, are performed parallel within the computer word in the single clock cycle. Equivalent to computer word there are basic registers available for holding a data unit in the computer. Figure 1 describes how the bit parallelism can be achieved in the computer.

With the use of bit parallelism in the field of String Matching, matching speed is improved up to certain level. But using such kinds of bit parallel based string matching, matching is quite difficult and having number of limitations [2]. These algorithms can be used in most of the real world applications [3] where string matching is required like as Intrusion Detection system, Plagiarism detection, Data Mining & Bioinformatics and etc.

Bit parallel string matching algorithms are faster than the other benchmark character matching based algorithms like as KMP [4], BM [5], BMH [6], BMHS [7], BMHS2 [8], BMI [9], Improved BMHS [10], Wu Manber [11] and Aho-Corasick [12] etc.

The Bit Parallel algorithms are based on the simulation of non-deterministic automata. It is simply the efficient simulation of non-deterministic automata. Figure 1 shows bit parallelism behaviour inside a computer.

Some of the important bit parallel string matching algorithms are Shift-OR [13], BNDM [14], TNDM [15], SBNDM [15], BNDMq [17], Shift-OR with Q-Gram [16], and Multiple pattern BNDM [18]. BNDM, TNDM, SBNDM and BNDMq are the single pattern string matching algorithms whereas Shift-OR, Shift-OR with Q-Grams and Multiple Patterns BNDM are the multiple pattern string matching algorithms.

This paper presents the working of above mentioned bit parallel string matching algorithms with examples. Here we elaborate the advantages and disadvantages of these algorithm and gives the bit parallel string matching history.

II. HISTORY OF BIT PARALLEL STRING MATCHING

In 1992 the Baeza–Yates and Gonnet gives the first bit parallel string matching algorithm named Shift-OR algorithm [13]. It was approximate multiple patterns string matching algorithm.

In 1998, based on Shift-OR algorithm Navarro and Raffinot proposed a new algorithm named as BNDM (Backward Non Deterministic Matching) [14]. BNDM is exact single pattern string matching algorithm. In 2003 another algorithm Simplified BNDM also known as SBNDM [15] was suggested. It was an enhanced version of BNDM. Here we do not require finding the longest prefix which will give the better performance in some cases.

In 2006 an enhanced edition of Shift-OR algorithm was launched by Salmela, Tarhio and Kytojoki known as Shift OR with Q-Gram [16]. It is also an approximate multiple...
string matching algorithm. It considers Q character at a
time for scanning.
In 2009 Branislav Durian, Jan Holub, Hannu Peltola and
Jorma Tarhio presented the concept of Q-gram in BNDM
algorithm known as BNDMq [17]. It scans the Q characters
at each arrangement. In 2010 Changsheng Miao, Guiran
Chang and Xingwei combine the concept of Q-gram with
BNDM and implemented Multiple Pattern BNDM
Algorithm [18]. Figure 2 shows the timeline history of bit
parallel string matching algorithms.

III. BIT PARALLEL STRING MATCHING ALGORITHMS

A. Shift-OR Algorithm:
Shift-OR [13] Algorithm gives the approximate matching
results it means algorithm matching results may contain
errors. In Shift OR algorithm searching is done from left to
right. But this is designed for working on patterns of equal
length plus the length less than or equal to the size of a
computer word.
Maximum multiple pattern algorithms are based on the
automata theory. They build automata of the patterns in
pre-processing phase. When the pattern set size is increased
then the size of automata also increases which is difficult to
maintain. Shift-OR algorithm is a replication of a
nondeterministic automata. It is suitable to be implemented
in hardware. Shift OR consist of two phases pre-processing
and searching.

Figure 2: Bit Parallel String Matching Timeline History

In pre-processing phase we find out the bit vector of the
every character which is possibly exists in the text. In this
ith corresponding bit is zero the same character appears at
position ‘i’ otherwise place 1 at ith position and write this in
reverse order. In searching phase automaton has a transition
from state ‘i’ to ‘i+1’ on the basis of input character say ‘c’.
If ith bit in B[c] is 0, and in state vector D where ith bit is
0 indicates that 0 occurs at MSB so we find the pattern at
that particular position. In Shift OR algorithm pre-
processing and searching both are simple. Here only Shift
and AND are used and no buffering is required. This
algorithm is working on the patterns whose lengths are
equal. Let’s take an example to understand Shift OR
algorithm.

Suppose King, thin and Apps be the patterns of length 4
and "DeepthinKing" be the text of length 12.
algorithm processing for this example. And the initial Bit
vector D is initialized to all 1’s.

Table 1: Shift-OR Algorithm Processing Example

<table>
<thead>
<tr>
<th>S.NO.</th>
<th>Matching Process</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scan &quot;D&quot;</td>
<td>D = (1110) OR (B[D]=1111)</td>
</tr>
<tr>
<td></td>
<td>D= 1111</td>
<td>Remain in the same state because MSB is not zero. D contains All 1’s.</td>
</tr>
<tr>
<td>2</td>
<td>Scan &quot;e&quot;</td>
<td>pos = pos+1=2</td>
</tr>
<tr>
<td></td>
<td>D= D&lt;&lt;1</td>
<td>Remain in the same state because MSB is not zero. D contains All 1’s.</td>
</tr>
<tr>
<td></td>
<td>D=(1110) OR (B[e]=1111)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=1111</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Scan &quot;e&quot;</td>
<td>pos = pos+1=3</td>
</tr>
<tr>
<td></td>
<td>D= D&lt;&lt;1</td>
<td>Remain in the same state because MSB is not zero. D contains All 1’s.</td>
</tr>
<tr>
<td></td>
<td>D=(1110) OR (B[e]=1111)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=1111</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Scan &quot;p&quot;</td>
<td>pos = pos+1=4</td>
</tr>
<tr>
<td></td>
<td>D= D&lt;&lt;1</td>
<td>Remain in the same state because MSB is not zero. D contains All 1’s.</td>
</tr>
<tr>
<td></td>
<td>D=(1110) OR (B[p]=1001)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=1111</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Scan &quot;t&quot;</td>
<td>pos = pos+1=5</td>
</tr>
<tr>
<td></td>
<td>D= D&lt;&lt;1</td>
<td>Move to the next state because D not contains All 1’s.</td>
</tr>
<tr>
<td></td>
<td>D=(1110) OR (B[t]=1110)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=1110</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Scan &quot;h&quot;</td>
<td>pos = pos+1=6</td>
</tr>
<tr>
<td></td>
<td>D= D&lt;&lt;1</td>
<td>Move to the next state because D not contains All 1’s.</td>
</tr>
<tr>
<td></td>
<td>D=(1100) OR (B[h]=1101)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=1101</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Scan &quot;r&quot;</td>
<td>pos = pos+1=7</td>
</tr>
<tr>
<td></td>
<td>D= D&lt;&lt;1</td>
<td>Move to the next state because D not contains All 1’s.</td>
</tr>
<tr>
<td></td>
<td>D=(1010) OR (B[r]=1001)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=1011</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Scan &quot;n&quot;</td>
<td>pos = pos+1=8</td>
</tr>
<tr>
<td></td>
<td>D= D&lt;&lt;1</td>
<td>MSB = 0 means patterns found at pos- m+1 i.e. 8-4-1=5.</td>
</tr>
<tr>
<td></td>
<td>D=(0110) OR (B[n]=0011)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=0111</td>
<td></td>
</tr>
</tbody>
</table>
B. BNDM Algorithm:
BNDM i.e. Backward Non Deterministic Matching [14] is single pattern exact string matching algorithm. In BNDM we scan the text from right to left respective of pattern. BNDM matching concept is based on shift OR Algorithm and suffix automata from BDM (Backward Deterministic Matching) Algorithm. This algorithm is simulation of BDM algorithm with the use of bit parallelism. BNDM simulates the non-deterministic version using bit parallelism.
BNDM comprises of two phases pre-processing and searching. In pre-processing phase we find Bit Vectors of each character that will possibly come in text. This is done by using the pattern characters. Bit Vectors are to be calculated by putting 1 for occurrence and 0 for non-occurrence of the character and we first take bit vector D with initial value i.e. all one. Here bit vector size is less than or equal to the word length of the computer. While searching, the pattern is searched with the help AND & SHIFT operations. Pattern is searched when MSB of D is 1 and value of character count pointer is 0.
For example Let Text T = ‘DeepthinKing’ and Pattern P = ‘King’. The Bit vectors of the characters are B [k] = 1000, B [i] = 0100, B [n] = 0010, B [g] = 0001 and other = 0000. Initial Value of D = 1111, pos = 0, j = 4 and last = 4. Table 2 shows the BNDM algorithm processing example.

Table 2 : BNDM Algorithm Processing Example

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Matching Process</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scan &quot;p&quot;</td>
<td>D = D &amp;B[Tpos + j] D= (D=1111)&amp;(B[p]=0000) D=0000. pos = pos +last =4 D=1111, j=4, last=4 D is all zero means mismatch with zero suffix entry.</td>
</tr>
<tr>
<td>2</td>
<td>Scan &quot;n&quot;</td>
<td>D = D &amp;B[Tpos + j] D= (D=1111)&amp;(B[n]=0010) D=0010. j= j-1= 3, D&lt;&lt;&lt; 1 i.e. 0100</td>
</tr>
<tr>
<td>3</td>
<td>Scan &quot;i&quot;</td>
<td>D = D &amp;B[Tpos + j] D= (D=1111)&amp;(B[i]=1000) D=1111. j= j-1=2, D&lt;&lt;&lt; 1 i.e. 1000</td>
</tr>
<tr>
<td>4</td>
<td>Scan &quot;h&quot;</td>
<td>D = D &amp;B[Tpos + j] D= (D=1111)&amp;(B[h]=0000) D=1000. pos = pos +last =8 D=1111, j=4, last=4</td>
</tr>
<tr>
<td>5</td>
<td>Scan &quot;g&quot;</td>
<td>D = D &amp;B[Tpos + j] D= (D=1111)&amp;(B[g]=0001) D=0001 j=j-1=3, D&lt;&lt;&lt; 1 i.e. 0010.</td>
</tr>
<tr>
<td>7</td>
<td>Scan &quot;i&quot;</td>
<td>D = D &amp;B[Tpos + j] D= (D=0010)&amp;(B[i]=1000) D=0100. j= j-1=1, D&lt;&lt;&lt; 1 i.e. 1000.</td>
</tr>
<tr>
<td>8</td>
<td>Scan &quot;K&quot;</td>
<td>D = D &amp;B[Tpos + j] D= (D=1000)&amp;(B[K]=1000) D=1000. j= j-1=0.</td>
</tr>
</tbody>
</table>
D. SBNDM Algorithm:
SBNDM [15] i.e. Simple Backward Non Deterministic Matching is same as BNDM algorithm. SBNDM is having some changes, for calculating the shifts. Because of that mechanism SBNDM algorithm is faster than the BNDM algorithm. Here we are not going to finding the largest prefix for shifting. Here we always go for first prefix. In SBNDM, the average length of shift is reduced. Here the innermost loop of algorithm becomes simple. It is faster than the BNDM algorithm in some of the favourable cases. Figure 4 shows the shifting logic of the SBNDM Algorithm.

E. BNDMq Algorithm:
BNDM with Q-gram [17] is an enhanced variation of the Basic BNDM algorithm. In BNDMq, the q-characters are read at each alignment before testing the state variable. In this algorithm loop has been made in order to quickly advance m-q+1 position, where m is the length of pattern. Here q can be varied according to our requirement. The whole algorithm can be easily understood with the help of the example given below.

Let us assume the text is "DeepthinKing" and pattern is "King". Bit vector B K = 1000, B i = 0100, B n = 0010, B g = 0001 and other = 0000.

F. Shift-OR with Q-Grams:
Shift OR with Q-gram [16] is an improved version of the Shift-OR algorithm. In Shift-OR with Q-gram algorithm we take Q character at a time for comparison by doing this, the size of simulation automata will be reduced and the number of comparisons for finding pattern is reduced up to certain level. The Q-gram can be Successive Q-gram or Overlie Q-gram. In Successive Q-gram we read pattern in a sequence of q character at a time while in Overlie Q-gram we take q character from each character of the patterns.

Let us take an example of Shift-OR Successive 2-Gram where Text is "DeepthinKing" and Patterns are 'inKing', and 'epthin'. Hence the Successive 2-gram of patterns are B in = 010, B Ki = 101, B ng = 011, B ep = 110, B th = 101 and other = 111. Table 4 shows the processing of the algorithm.

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Matching Process</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D=111,D&lt;&lt;1</td>
<td>No match retain the same state</td>
</tr>
<tr>
<td></td>
<td>D=((D=110)OR(B[De]=111))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=111</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>D=111,D&lt;&lt;1</td>
<td>No match retain the same state</td>
</tr>
<tr>
<td></td>
<td>D=((D=110)OR(B[se]=111))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=111</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>D=111,D&lt;&lt;1</td>
<td>Move to next state</td>
</tr>
<tr>
<td></td>
<td>D=((D=110)OR(B[ep]=110))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=110</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>D=110,D&lt;&lt;1</td>
<td>Move to next state</td>
</tr>
<tr>
<td></td>
<td>D=((D=100)OR(B[th]=101))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=101</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D=101,D&lt;&lt;1</td>
<td>Move to next state</td>
</tr>
<tr>
<td></td>
<td>D=((D=010)OR(B[ni]=010))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=010</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>D=101,D&lt;&lt;1</td>
<td>Move to next state</td>
</tr>
<tr>
<td></td>
<td>D=((D=100)OR(B[Ki]=101))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=101</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>D=101,D&lt;&lt;1</td>
<td>Move to next state</td>
</tr>
<tr>
<td></td>
<td>D=((D=010)OR(B[ng]=011))</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D=011</td>
<td></td>
</tr>
</tbody>
</table>

MSB 1 means pattern found.
Because of the Q-Grams technique Shift-OR Algorithm becomes faster and the total number of false matches is reduced.

**G. Multiple Patterns BNDM Algorithm:**
Changsheng Miao, Guiran Chang and Xingwei Wang convert the BNDM algorithm in multiple pattern BNDM algorithms [17]. They develop Filtering Based Multiple String Matching Algorithm by Combining Q-Grams and BNDM. It is very simple algorithm and may contain false matches. In multiple pattern BNDM string matching algorithm multiple patterns can be searched this was not possible in normal BNDM algorithm. We need filtering after pattern match to find the exact match which will take extra time to search. This algorithm is the similar to Shift-OR with Q-gram only the change is instead of Shift-OR it uses the BNDM matching process.

**IV. CONCLUSION**
Bit parallelism is the inherent behaviour of the computer through which we can get the speed-up in various real world applications where bitwise operations are used. Bit parallel string matching algorithms are the new series of efficient algorithms. BNDM, TNDM, SBNDM, BNDMq are the bit parallel single pattern string matching algorithms and Shift-OR, Shift-OR with Q-Grams, multiple pattern BNDM are used for multiple pattern matching bit parallel algorithms. These are efficient while comparing to standard character based algorithm. These algorithms have some disadvantages, major disadvantage of these algorithm is working on patterns of length less than or equal to computer word. Multiple pattern algorithms were working on the pattern of equal length and it allows the approximation in matching results.

**V. FUTURE WORK**
Multiple pattern bit parallel exact string matching algorithm is not available yet. So multiple pattern exact string matching is required for matching. Word size limitation present in the bit parallel algorithm can also be removed in future.

**REFERENCES**
[17] Branislav Durian, Jan Holub, Hannu Peltoila and Jorma Tarhio,”Tuning BNDM with q-grams”, In the proc. Of workshop on algorithm engineering and experiments, SIAM USA, pp. 29-37, 2009.