Multiple-Pattern Matching Using Improved Bit-Parallel Approach

K. H. Chen, G. S. Huang and R. C. T. Lee*

Department of Computer Science and Information Engineering,
National Chi Nan University, Puli, Nantou, Taiwan.
{s95321902, shieng, rctlee}@ncnu.edu.tw

Abstract

We consider a fundamental problem testing if every bit is zero in a bit vector in a so-called bit-parallel machine. In this machine, a machine word can have unlimited number of bits, and logical operations such as bitwise-and, bitwise-or, and so on, can be done in $O(1)$ time. However, in each time, only one bit in a bit vector can be examined. We show how to solve this fundamental problem in $O(1)$ time by using the composition of bitwise logical operations and basic arithmetic operations, and apply this technique to solve the multiple-pattern matching problem. Multiple-pattern matching is the problem of finding all occurrences of patterns in a text. There are bit-parallel algorithm Shift-And [11] that can solve it. In this paper, we show how to improve the performance of the Shift-And algorithm, and accordingly, to the performance of multiple-pattern matching under the bit-parallel machine we have addressed. Our idea is based on a key lemma from Ou and Lee [12].

1 Introduction

We consider the multiple fixed length pattern matching problem: Given a set P of r patterns $p_1, p_2, \cdots, p_r$ with lengths m respectively and a text T with length n, find all occurrences of p_i, $1 \leq i \leq r$, in T. Let Σ be the constant-sized alphabet. The first algorithm which solved this problem was the Aho-Corasick in $O(n)$ time (AC algorithm for short) [1]. It creates a finite state machine from the set of patterns P, scans each character of T and conducts a trie searching. Based on this approach, many studies combined it with other single-pattern matching algorithms. Commentz-Walter [4] combined the idea of the Boyer-Moore algorithm [3] and the AC algorithm to solve the problem. Crochemore et al.[6] integrated the ideas of the Reverse Factor algorithm [5] to the AC algorithm. It employed two automata-theoretic tools: an AC machine and a suffix automaton. Later, Raffinot developed similar algorithm, called MultiDBM [13]. Besides the automata approach, for multiple patterns matching, many researchers used "hash" technique to find possible patterns which occur in [7, 15].

Some researchers used bit parallelism to represent patterns and text. In single pattern matching, the bit parallel approach is very useful. Many papers used bit parallel technique to solve this problem [2, 11]. In [2], Shift-Or operation and KMP Algorithm were combined. In [8], bit-parallel approach was combined with Sunday's quick searching algorithm [14]. The Shift-And Algorithm [11] combined bit parallel approach with the Reverse Factor Algorithm [5], but their algorithm has some flaws which we will discuss later.

Our algorithm is quite similar to the Shift-And Algorithm. However, we used the logical operations first mentioned in [10] and soundly developed in [12].

The organization of this paper is as follows. In Sect. 2, we briefly introduce the basic idea of the Shift-And algorithm for single pattern case, and then propose the all zero vector problem arising from designing Shift-And algorithms for multiple pattern matching in Sect. 3. Logical operations to solve the all zero vector problem are in Sect. 4. We also provide a complete example of the improved Shift-And algorithm in Sect. 5. Finally, we conclude this work in Sect. 6.

2 The Reverse Factor Algorithm Solved by the Shift-And Technique

A comprehensive introduction of the Reverse Factor Algorithm can be found in [9]. As pointed out in [9], the essential point in the algorithm is to find the longest suffix of a window W of T which is equal to a prefix of X of length m, denoted as $LSP(W, X)$.

To use the bit parallel approach, we first define an incidence vector B with length m. $B[a]$ is set 1 at i-th position if and only if $x_i = a$. For example, let $X = gtcaa$. Then its mask $B[a] = (00011)$, $B[c] = (00100)$, $B[t] = (01000)$ and $B[g] = (10000)$. Besides, the algorithm has a bit vector $D = d_1 \cdots d_m$ where the bit d_i after k iterations is set 1 if and only if $x_i \cdots x_{i+k-1} = w_{m-k+1} \cdots w_m$. Initially, we initialize $d_i = 1$, $1 \leq i \leq m$.

To find $LSP(W, X)$, we scan the window from right to left. Let $last$ be finding the latest a suffix of W which is equal to a prefix of X. When a new character

*To whom correspondence should be addressed. Email:rctlee@ncnu.edu.tw in the window was scanned at the k-th iteration, we update D. The formula to update D is as follows

$$D' \leftarrow (D \ \& \ B[w_{m-k+1}]) \tag{1}$$

If every bit is 0 in D, we stop. Then $|LSP(W,X)| = last$ if $last \neq 0$; otherwise, $|LSP(W,X)| = 0$. The shift $S_s = m - |LSP(W,X)|$. If we can perform m iterations and $d_1 = 1$, then we stop and report a match. A proper suffix of a string is a suffix of the string, but not equal to the string. Let $|LPSP(W,X)|$ be the length of the longest proper suffix of W which is equal to a prefix of X. Then $|LPSP(W,X)| = last$ if $last \neq 0$; otherwise, $|LPSP(W,X)| = 0$. In this case, the shift $S_s = m - |LPSP(W,X)|$.

If $D \neq 0^m$ and $k < m$, we must check whether $d_1 = 1$. That is, set $last$ to be k. It means that we have found a suffix of W which is equal to the prefix of X with length k. Otherwise, we shift vector D one bit to the left and fill the vacant bit to be 0. For instance, if $D = (0110)$, after shifting, we have $D' = (1100)$. This is denoted as

$$D' \leftarrow D << 1 \tag{2}$$

Algorithm 1 shows a complete description of the single-pattern matching using the Shift-And algorithm.

Algorithm 1: The single-pattern matching using the Shift-And algorithm

```
Input: Two strings $T$ and $X$ with length $n$ and $m$
       respectively
Output: The locations of all occurrences of $X$ in $T$
begin
    $|LPSP(W,X)|$ is the length of the longest
    proper suffix of the window in $T$ which is equal
    to a prefix of $X$.
    $|LSP(W,X)|$ is the length of the longest suffix of
    the window in $T$ which is equal to a prefix of $X$.
    Set $i = m$.
    Compute all of the incidence vectors $B$ of $X$.
    While $i \leq n$ do
      $k = 0$ and $last = 0$.
      $D = 1^m$.
      $W = t_{i-m}t_{i-m+1}\cdots t_i$.
      $|LPSP(W,X)| = 0$.
      While (1) do
          $k \leftarrow k + 1$.
          $D \leftarrow D \ \& \ B[w_{m-k+1}]$.
          If $D = 0^m$, then $|LSP(W,X)| = last$ and exit.
          If $k = m$ and $d_1 = 1$, report an occurrence of
              $X$ at $t_{i-m}$, $|LPSP(W,X)| = last$ and exit.
          If $d_1 = 1$, then $last \leftarrow k$.
          $D \leftarrow D << 1$.
      If $D \neq 0^m$, then $i \leftarrow i + (m - |LPSP(W,X)|)$;
          otherwise, $i \leftarrow i + (m - |LSP(W,X)|)$.
end
```

We give an example where it is assumed that $X = actg$, $T = ttcgacgt$. We begin with

$$T = ttcgacgt, D = (1111), m = 4, n = 9, last = 0,$$
$$B[a] = (1000),$$
$$B[c] = (0100),$$
$$B[g] = (0010),$$
$$B[t] = (0001).$$

The first window is $W = ttcg$. We found $D = 0^m$ and $last = 0$ in 3-th iteration. Therefore, $|LSP(W,X)| = 0$. We shift the window $S_s = m - |LSP(W,X)| = 4 - 0 = 4$. In the second attempt, we scan $W = actg$. In 4-th iteration, because $k = 4 = m$ and $last = 0$, we stop and report a match. Then $|LPSP(W,X) = 0|$. $S_s = m - |LPSP(W,X)| = 4 - 0 = 4$. The processes of two attempts are shown in Figure 1 and Figure 2 respectively.

3 The All Zero Vector Problem Arising from Designing Bit-Parallel Algorithms for Multiple Pattern Matching

For multiple-patterns matching, its preprocessing phase is like the single-pattern matching. Let the set of patterns be $P = \{p_1, p_2, \cdots, p_r\}$. We concatenate the patterns as follows: $S = p_1p_2\cdots p_r$. For example, let $P = \{cct, aca, gtc\}$. Then the concatenation of $S = cctacagtc$.

Let us assume that we have $T = accttac$. For the set of patterns, the length of each pattern is 3. We therefore set the window to be $W = T(1,3) = acc$ with length 3. For each pattern p_i in $P = \{cct, aca, gtc\}$, we now try to find $LSP(W, p_i)$. It can be seen that $|LSP(W,p_1)| = 2$, $|LSP(W,p_2)| = 0$ and $|LSP(W,p_3)| = 0$. To find this in a bit-parallel manner, let M be the total length of the patterns. We further define an incidence vector B with length $|S|$. $B[a]$ is set 1 at i-th position if and only if $s_i = a$. For our example where $S = cctacagtc$,

$$B[a] = (000101000),$$
$$B[c] = (110010001),$$
$$B[g] = (001000010),$$
$$B[t] = (000000100).$$

Let bit vector D consist of all 1's to start with. Since our first window is $W = T(1,3) = acc$ and the rightmost character w_3 is c, we perform an AND operation of $B[c]$ and D. We obtain $D = (110010001)$ which indicates c appears in locations 1, 2, 5 and 9 in S. At this point, we have to examine whether D contains at least one 1 because if D contains only 0, there exists no substring of p_i for any i which appears in W and we may stop the process. Therefore, we must solve the following All Zero Vector Problem: Given a vector X, determine whether X contains all 0's. The algorithm of Navarro and Raffinot [11] checks all bits in $O(M)$ time which makes that algorithm not efficient.

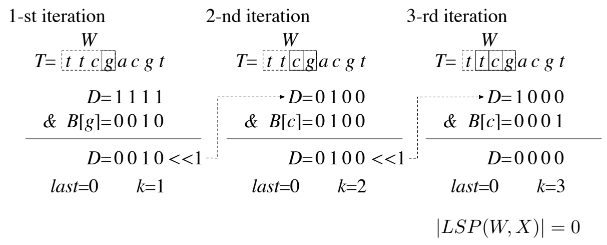

Figure 1: The first attempt.

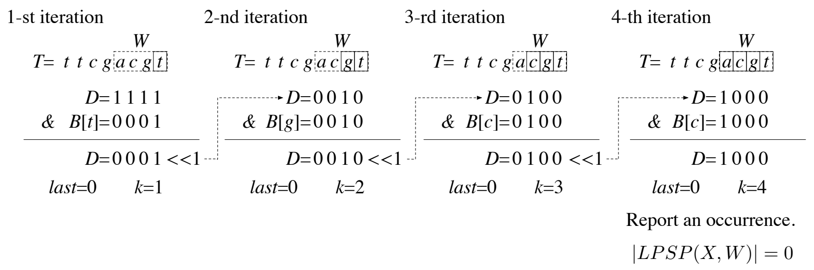

Figure 2: The second attempt.

We shall point out that we may use the bit-parallel logical operation in [12] to solve the problem in $O(1)$ time. We will discuss this operation in the next section.

There is another instance where we shall encounter the All Zero Vector Problem. If D contains at least 1, it indicates that c does appear in one of the p_i's. But we actually need to know whether it appears as the first character of some p_i. We use a mask $A = a_1a_2 \cdots a_m$ where a_i is set 1 if i-th position in S is the first character of a pattern, otherwise, a_i is set 0. We then perform an AND operation of A and D to produce a vector C. If C contains only 0, there does not exist any $LSP(W, p_i)$. For the previous case, $A = (100100100)$. $C = A \ \& \ D = (100100100)\&(110010001) = (100000000)$. Vector C indicates only the first character of p_1 is c. As shown above, we have to examine whether C contains all 0's. Thus, we have to solve another All Zero Vector Problem here.

4 Logical Operations to Solve the All Zero Vector Problem

Let $X = (x_1x_2 \cdots x_m)$ be a vector consisting of 1's and 0's. In the All Zero Vector problem, we want to check whether $x_i = 0$ for $1 \leq i \leq M$ using only logical operations, such as AND and OR etc. By modifying slightly the results in [12], we can use the following formula.

$$Y \equiv (((\overline{X} + 0^{M-1}1) \oplus \overline{X}) \ \& \ \overline{X}) \qquad (3)$$

In Y, $y_1 = 1$ if and only if $x_i = 0$ for $1 \leq i \leq M$. Here, $\overline{X}$ is the complement vector of X and the operation '+' is the binary addition from right to left.

For example, assume that $X = (000000)$. We have $\overline{X} = (111111)$. Then $Y = (((111111) + (000001)) \oplus (111111)) \ \& \ (111111) = (11111)$ and $y_1 = 1$.

For another example, assume that $X = (001010)$. We have $\overline{X} = (110101)$. Then $Y = (((110101) + (000001)) \oplus (110101)) \ \& \ (110101) = (000001)$ and $y_1 = 0$.

For the correctness of this approach, consult [12].

5 The Improved Shift-And Algorithm

Based on Equation (3), we can easy to examine whether the bits of D are all 0's. Recall our case that $D = (110010001)$. We have $\overline{D} = (001101110)$. Then

$$\begin{aligned} E &= (((\overline{D} + 0^8 1) \oplus \overline{D}) \ \& \ \overline{D}) \\ &= (((001101110) + (000000001)) \\ &\quad \oplus (001101110)) \ \& \ (001101110) \\ &= (000000001). \end{aligned}$$

Notice that $e_1 = 0$, which means that there exists at least one bit which is 1 in D. We further examine whether it appears as the first character of some p_i. We have $C = A \ \& \ D = (100100100) \ \& \ (110010001) = (100000000)$ and $\overline{C} = (011111111)$. Then

$$\begin{aligned} F &= (((\overline{C} + 0^8 1) \oplus \overline{C}) \ \& \ \overline{C}) \\ &= (((011111111) + (000000001)) \\ &\quad \oplus (01111111)) \ \& \ (01111111) \\ &= (01111111). \end{aligned}$$

Notice that $f_1 = 0$ indicates that the bits in C are not all 0's. It means that we have found a suffix of

W which is equal to a prefix of p_i with length 1. We set *last* to 1. If we can perform m iterations and also find $f_1 = 0$, then we report the existence of an exact match and stop the process. Let $|LPSP(W, p_i)|$ be the length of the longest proper suffix of W which is equal to a prefix of p_i. Then $|LPSP(W, p_i)| = last$ if $last \neq 0$; otherwise, $|LPSP(W, p_i)| = 0$.

If every bit in D is 0, we stop. Hence $|LSP(W, p_i)| = last$ if $last \neq 0$; otherwise, $|LSP(W, p_i) = 0|$.

If $D \neq 0^m$, we shift vector D one bit left for all patterns and fill their vacant bits to be 0. For instance, $D = (110010001)$, $A = (100100100)$. Then $\overline{A} = (011011011)$. We perform an AND operation of D and $\overline{A}$. We obtain (010010001). After shifting one bit, we have $D' = (100100010)$. This is denoted as

$$D' \leftarrow (D \ \& \ \overline{A}) << 1 \qquad (4)$$

abcdef

See algorithm 2 for a complete description of the improved Shift-And algorithm.

```
Algorithm 2: The improved Shift-And algorithm
  Input: A text $T$ of length $n$ and a set $P$ of $r$
        patterns $\{p_1, p_2, \cdots, p_r\}$ with length $m$.
  Output: The locations of all occurrences in $T$
  begin
      $|LPSP(W, p_i)|$ is the length of the longest proper
      suffix of the window in $T$ which is equal to a
      prefix of $p_i$.
      $|LSP(W, p_i)|$ is the length of the longest suffix of
      the window in $T$ which is equal to a prefix of $p_i$.
      $k$ represents the $k$-th iteration.
      $M = \sum_{i=1}^{r} m$.
      Set $S = p_1 p_2 \cdots p_r$ and the mask
      $A = a_1 a_2 \cdots a_M$ where $a_i = 1$ if $i$-th position is
      the first character of a pattern.
      Compute all of the incidence vectors $B$ of $S$.
      Set $i = m$.
      While $i \leq n$ do
        $k = 0$ and $last = 0$.
        $D = 1^M$.
        $W = t_{i-m} t_{i-m+1} \cdots t_i$.
        $|LPSP(W, p_i)| = 0$.
        While (1) do
            $k \leftarrow k + 1$.
            $D \leftarrow D \ \& \ B[w_{m-k+1}]$.
            $E \leftarrow ((1 + \overline{D}) \oplus \overline{D}) \ \& \ \overline{D}$.
            If $e_1 = 1$, then $|LSP(W, p_i)| = last$ and exit.
            $C \leftarrow D \ \& \ A$.
            $F \leftarrow ((1 + \overline{C}) \oplus \overline{C}) \ \& \ \overline{C}$.
            If $f_1 = 0$ and $k = m$, report an occurrence,
                $|LPSP(W, p_i)| = last$ and exit.
            If $f_1 = 0$, then $last \leftarrow k$.
            $D \leftarrow (D \ \& \ \overline{A}) << 1$.
        If $D \neq 0^m$, then $i \leftarrow i + (m - |LPSP(W, p_i)|)$;
            otherwise, $i \leftarrow i + (m - |LSP(W, p_i)|)$.
  end
```

$$\begin{array}{rcl} S & = & cctacagtc, \\ A & = & (100100100), \\ B[a] & = & (000101000), \\ B[c] & = & (110010001), \\ B[t] & = & (001000010), \\ B[g] & = & (000000100), \end{array}$$

Figure 3: The matrix B and the filter vector A.

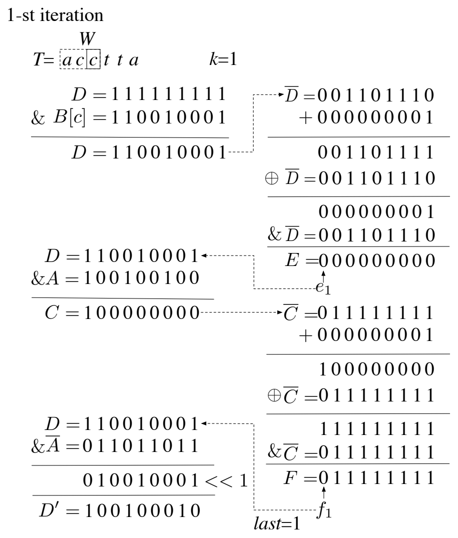

Figure 4: The first iteration in the first attempt.

We give a complete example that $T = acctta$, $m = 3$ and $P = cct, aca, gtc$. The concatenation of $S = cctacagtc$. In the preprocessing phase, the incidence vectors B of S and the mask A are shown in Figure 3.

We scan the characters of window $W = acc$ from right to left. The rightmost character w_3 is c and the first iteration is shown in Figure 4.

In Figure 4, we found that $e_1 = 0$ which indicates the bits in D are not all 0's. We further examine whether $f_1 = 0$. We obtain $f_1 = 0$, which means there exists a suffix of p_i which is equal to a prefix of W with length 1. We set $last = 1$.

In the second iteration, we have $D = (100100010)$ and the character w_2 is c. We found that $e_1 = 0$ and $f_1 = 0$, which indicates there exists a suffix of p_1 which is equal to a prefix of W with length 2. We set $last = 2$. The process of the 2-nd iteration is shown in Figure 5.

In the third iteration, we have $D = (00000000)$ and the character w_1 is a. After computing, we found that $e_1 = 1$. We stop the process. Note that $|LSP(W, p_i)| = last = 2$. Then the shift $S_m = 3 - 2 = 1$. The process of the 3-rd iteration is shown in Figure 6.

In the second attempt, we scan the characters of window $W = cct$ from right to left. We start with the rightmost character $w_3 = t$. The process for the first iteration in the second attempt is shown in Figure 7.

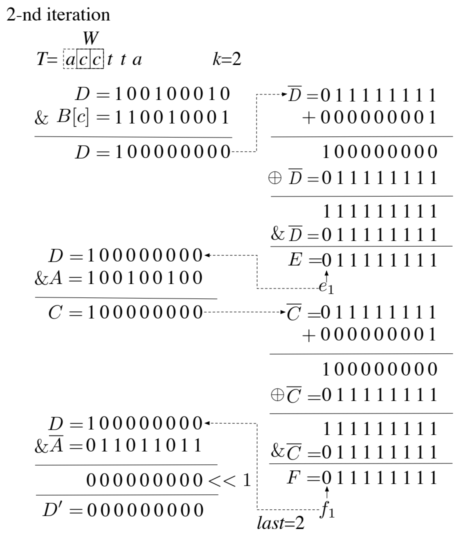

Figure 5: The second iteration in the first attempt.

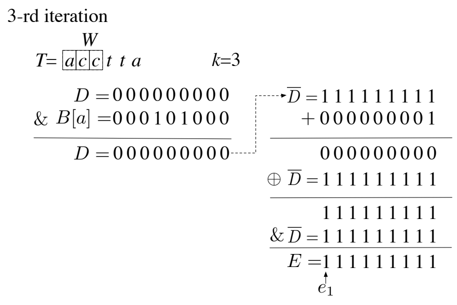

Figure 6: The third iteration in the first attempt.

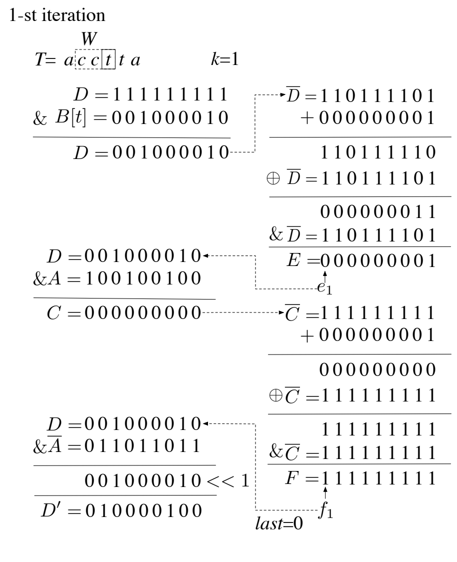

Figure 7: The first iteration in the second attempt.

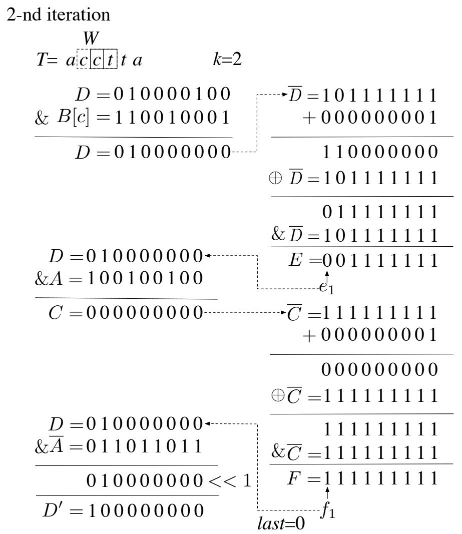

Figure 8: The second iteration in the second attempt.

In Figure 7, we found that $e_1 = 0$ and $f_1 = 1$, which indicates it is no $LSP(W, p_i)$. We shift D one bit left based on Equation (4).

In the second iteration, we have $D = (010000100)$ and the character w_2 is c. We found that $e_1 = 0$ and $f_1 = 1$. We shift D one bit left and get $D = (100000000)$. The result is shown in Figure 8.

In the third iteration, we have $D = (100000000)$ and the character w_1 is c. We found that $e_1 = 0$, $f_1 = 0$ and $k = m = 3$. We report the existence of an exact match and set $|LPSP(W, p_i) = 0|$. Then the shift $S_m = 3 - 0 = 3$, which is shown in Figure 9.

After shifting, the boundary of the window is larger than the length of T. The searching process has finished.

6 Conclusions

In this paper, we show how to solve the all-zero-vector problem in a bit-parallel machine in $O(1)$ time. Based on this breakthrough, we successfully improve the performance of the Shift-And algorithm in solving the multiple-pattern matching problem. A by-product of this achievement is to solve the single pattern matching problem, which is simply a special case of the multiple-pattern matching problem. At the end of this paper, we put some words to justify the modeling of the bit-parallel machines. One might wonder why we put restriction on the way to examine the content of a bit-vector one bit in each time. In fact, this idea is inspired from quantum computing. In quantum computing, a quantum bit (qubit) can actually represent many states through a probability distribution at the same time. A qubit can be manipulated as a single unit, and qubits can interact with one another efficiently (e.g., the superposition and entanglement). Moreover, the multi-state information is still

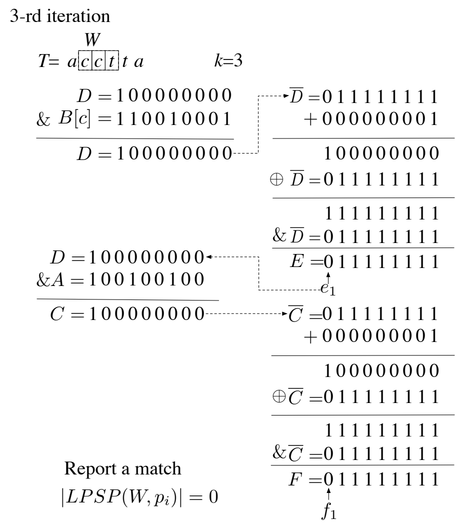

Figure 9: The third iteration in the second attempt.

kept, as long as one does not examine the content of a qubit. However, the content of a quantum bit quickly becomes fixed when one tries to examine it. The key point is that this magical ability is gifted to data items. As a contrast, the observers are still ordinary and usual. Applying this idea to the bit-parallel machine, we therefore assume all bitwise-like operations can be processed in parallel, and thus they can be done in $O(1)$ time. However, when we try to examine the content of a bit-vector, we do not the ability to examine all bits at the same time since there are unlimited number of bits inside it. We will explore more on this interesting model for computation in the future. We remark that one can simulate the effect of a bit-vector of a bit-parallel machine by the traditional Random Access Machine (RAM). It would take $O(M/w)$ time if the length of the bit-vector is M and a machine word has w bits.

Acknowledgment

The authors would like to thank C. S. Ou for his generous and enthusiastic discussions on Sect. 4.

References

[1] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[2] R. A. Baeza-Yates and G. H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, 1992.

[3] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[4] B. Commentz-Walter. A string matching algorithm fast on the average. In *Proceedings of the 6th Colloquium, on Automata, Languages and Programming*, pages 118–132, London, UK, 1979. Springer-Verlag.

[5] M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter. Speeding up two string-matching algorithms. *Algorithmica*, 12(4-5):247–267, 1994.

[6] M. Crochemore, A. Czumaj, L. Gasieniec, T. Lecroq, W. Plandowski, and W. Rytter. Fast practical multi-pattern matching. *Information Processing Letters*, 71(3-4):107 – 113, 1999.

[7] S. Kim and Y. Kim. A fast multiple string-pattern matching algorithm. In *In Proceedings of 17th AoM/IAoM Conference on Computer Science*, 1999.

[8] M. O. Külekci. Tara: An algorithm for fast searching of multiple patterns on text files. In *Computer and information sciences, 2007. iscis 2007. 22nd international symposium on*, pages 1–6, Nov. 2007.

[9] R. C. T. Lee. String matching. *(Unpublished)*, 2010.

[10] G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, 46:1–13, 1999.

[11] G. Navarro and M. Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching, LNCS v. 1448*, pages 14–33, 1998.

[12] C. S. Ou and R. C. T. Lee. Bit-parallel operations to investigate properties of logical vectors by logical operations. *(Unpublished)*, 2010.

[13] M. Raffinot. On the multi backward dawg matching algorithm (multibdm). In *In: R. Baeza-Yates, Editor, Proc. 4th South American Workshop on String Processing, Carleton University Press*, pages 149–165, 1997.

[14] D. M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, 33(8):132–142, 1990.

[15] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical report, Department of Computer Science, Chung-Cheng University, 1994.