Journal of Theoretical and Applied Information Technology

www.jatit.org

AN EFFICIENT ALGORITHM FOR MULTIPLICATION BASED ON DNA COMPUTING

[1] **Santosh Kumar Mishra**, [2] **Sanchita Paul**

[1] Student (M.Tech), Deptt. of Computer Science & Engineering, B.I.T. Mesra, Ranchi, India-835215

[2] Lecturer, Deptt. of Computer Science & Engineering, B.I.T., Mesra, Ranchi, India-835215

E-mail:-santosh_mta@yahoo.com, sanchita07@gmail.com

ABSTRACT

DNA Computing utilizes the properties of DNA for performing the computations. The computations include arithmetic and logical operations such as multiplication. We first show a procedure for multiplication of a pair of two binary numbers. The procedure mainly consist of bit-shift operation where the operation depends on bit position of one's (ignoring zero's) in the multiplicand and finally addition operations which take place simultaneously in each steps. The above method takes O(1) time in the best case which exists when each bit of multiplicand is zero. However, the time complexity of proposed algorithm is O(n) for average and worst case and the space complexity of proposed algorithm is O(n) for average case, worst case and best case. In addition the most merit of this model is simple coding and its time efficiency.

Keywords: DNA computing, DNA computation Model, Multiplication.

1. INTRODUCTION

DNA computing is new computation paradigms, which proposes the use of molecular biology tools to solve different mathematical problems. It is a form of computing which use DNA, biochemistry and molecular biology, instead of the traditional silicon-based computer technologies. The primary advantage of DNA based computation is the ability to handle millions of operations in parallel. DNA computing is fundamentally similar to parallel computing in that it takes advantage of the many different molecules of DNA to try many different possibilities at once.

DNA computing has two important features, which are Watson-Crick complimentarily and massive parallelism. Using the features, we solve some optimization problems, which usually need exponential time on silicon-based computers, in polynomial steps with DNA molecules. However, for DNA computing to be applicable on a various range of problems for primitive operations, such as logic or arithmetic operations. A number of procedures have been proposed for the primitive operations with DNA molecules [1,3,4,10,11,12,14]. One procedure was proposed by Gaurnieri et.al. [12] for addition of two binary numbers of m bits .The procedure uses O (m) steps using O(m) different DNA strands .

Another model which allows for representing and manipulating binary numbers on DNA chip by H.Hug and Rescuer [4] applies parallel execution of primitive operations. Kamino et.al. [14] proposed a procedure which computes maximum n binary numbers of m bits, which runs in O(1) steps using O(mn^2) DNA strands. A.fujiwara [1] shows one procedure for multiplication and division in DNA computing. This method shows multiplication of two binary numbers of m bits in O(log m) steps using O(m^2) DNA strands.

In this paper, we present a procedure for multiplication of a pair of n bit binary numbers. This procedure involves two operation first is bit-shift operation depending on bit position of one's (ignoring zero's) in the multiplicand and finally addition operations which take place simultaneously in each steps. The above method takes O(1) time in the best case using O(n) DNA strands which exists when each bit of multiplicand is zero. However, the time complexity of proposed algorithm is O(n) using O(n^2) DNA strands and the space complexity of proposed algorithm is O(n) for average case, worst case and best case.

2. PROCEDURE FOR MULTIPLICATION

In this section, we have described a method for multiplication of a pair of n bit binary numbers. The method for multiplication mainly consists of bit-shift and addition operations according to position of each bit in multiplicand.

Q=Multiplicand, where we check the bit position of 1's;

M= Multiplier;

A= Temporary Register, where bit-shift operations performed and stored as temporary;

S= Partial Result; where temporary result of A (after bit-shift operations) added to S;

R= Where, bit positions of Q (multiplicand) containing 1;

M<<R[i] = M shifted R[i] times;

It consists of following basic operation

- ? if the value of i-th bit of Q is 1, we store the j left-shifted value of M in an address A .
- ? If the value of i-th bit of Q is 0, there is no operation.
- ? The results of operation stored in S.

S=S+A	A=M<<R[i]	Q	M	R[i]	Remarks
00000000	00001011	1101	1011	3,2,0	Initialization
00001011	00001011	1101	1011	i=0 R[0]=0	No shift Operation on M as R[i]=0
00001011 00101100 00110111	00001011 00010110 00101100	1101	1011	i=1 R[1]=2	2 bit Left Shift on M as R[i]=2
00110111 01011000 10001111	00001011 00010110 00101100 01011000	1101	1011	i=2 R[2]=3	3 bit Left Shift on M as R[i]=3.
10001111					RESULT

Figure 1: [bit-shift and addition operation]

2.1 FLOW CHART

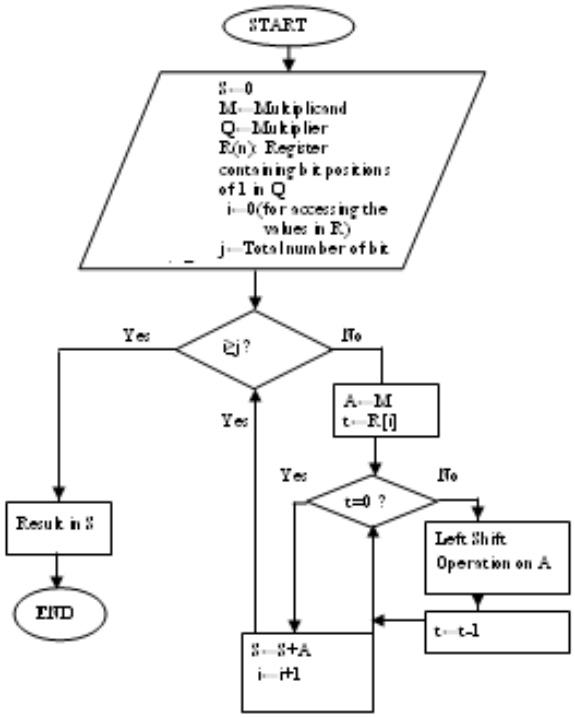

Figure 2: Flowchart for multiplication

2.2 ALGORITHM

1. Initialize Registers.

S=0 //sum register used to perform addition (S=S+A). [It takes O(1) complexity].

A=Auxiliary register for storing the result of bit-shift operation. [It takes O(1) complexity].

Q=Multiplier. [It takes O(1) complexity].

M=Multiplicand. [it takes O(1) complexity].

R //processing register of size n that contains the bit-positions of 1 in Q. [It takes O(1) complexity].

i=0 // for accessing values in Register R. [It takes O(1) complexity].

j=k-1 // Number of Values stored in R. [it takes O(1) complexity].

2. Repeat step 3-4 until i≥j

3. a. A=M.

b. t=R[i].

3c.Until (t>0) repeat step (i) & (ii). Repeataion takes O(n) complexity.

i). Circular left shift A.

ii). t=t-1

4. S=S+A

i=i+1

5. Final result in register S. //it takes O(1) complexity.

Total time complexity: O(n)

3 PRELIMINARIES

3.1 DNA STRANDS

DNA strands are a basic element for DNA computing. DNA strands are similar to a memory module used on a silicon-based computer. A single strand of DNA is defined as a string of four different base nucleotides (A, T, C, G). While DNA strands can be synthesized using biological method [9,11], we assume that each single strands of DNA represent a symbol over a finite alphabet Σ(sigma). DNA computing has main concept of Watson-Crick complementary method (complementary base pairs-Adenine and Thymine, Guanine & Cytosine) and massive-parallelism. We can solve some problems in a minimum number of steps with biological DNA molecules by help of these concepts. We describe the alphabet Σ={e_0, e_1,?e_{m-1}, $\bar{e}_0$, $\bar{e}_1$?.. $\bar{e}_{m-1}$}, where the symbol e_i and $\bar{e}_i$ (0<=i<=m-1) are complements. A single strand is a sequence of one of one or more symbol in Σ. Two single strands of DNA form a double strand if and only if the single strands are complements of each other. A double strand with e_i, $\bar{e}_i$ is denoted by $\binom{e_i}{\bar{e}_i}$.

The DNA strands which are single or double both are stored in a test tube .The following expressions denotes test tubes T_1 and T_2, which store single and double DNA strands. For example, $T_1=\{e_0e_1, \overline{e_0e_1}\}$ denotes two single strands are stored in a test tube T_1. The set of strands stored in a test tube is allowed to be a k-multiset. Where, each strand has k-copies in the test tube and k-depends on the error involved in DNA manipulations. We describe each strand in a set represented by test tube as one unit. For example, the test tube T_1 contains two units of e_0e_1, and three units of $\bar{e}_0\bar{e}_1$, represented as $T_1=\{e_0e_1,e_0e_1, \bar{e}_0\bar{e}_1,\bar{e}_0\bar{e}_1,\bar{e}_0\bar{e}_1\}$.

3.2 COMPUTATIONAL MODEL FOR DNA COMPUTING

Several theoretical and mathematical computational models have been proposed for DNA computing [2, 3, 4, 5, 6, 7, 8]. The computational model used in this paper is same as [11,3]. In this section, we introduce this computational model. The DNA may be single or double stranded. A single strand of DNA is defined as a string of symbols over a finite alphabet Σ(sigma).

We define these alphabet as Σ={e_0 e_1??..e_{m-1} $\bar{e}_0$ $\bar{e}_1$? $\bar{e}_{m-1}$}.Where the symbols e_i, $\bar{e}_i$(0<i<m-1) are complement.. A double strand with e_i, $\bar{e}_i$ is denoted by $\{e_i, \bar{e}_i\}$. The model allows the following seven DNA manipulations which are widely used in DNA computing:

[1]. Merge: Given two test tubes T_1, T_2. Merge (T_1, T_2) stores the union in a single test tube $T_1 \cup T_2$ in T_1.

[2] Copy: Given test tube T_1, a new test tube T_2 can be produced with same contents using the manipulation operation Copy (T_1, T_2).

[3]. Detect: Detects whether the test tube T contains at least one strand. Detect (T) produces the output "Yes" if T contains DNA strands, otherwise returns "No".

[4]. Separation: Given a test tube T_1 and a set of strings X, separation (T_1,X, T_2) removes all single strands containing a string in X from T_1, and produces a test tube T_2 with the removed strands.

[5]Cleavage: Given a test tube T and string of two symbols e_0e_1, the operation Cleavage (T, e_0e_1) cuts each double strand containing

$\binom{e_0e_1}{\overline{e_0e_1}}$

$\binom{\alpha_0 e_0e_1\beta_0\alpha}{\alpha_1 e_0e_1\beta_1} \Rightarrow \binom{\alpha_0e_0}{\alpha_1e_0}, \binom{e_1\beta_0}{e_1\beta_1}$

It is assumed that Cleavage can only be applied to some specified symbols over the alphabetΣ.

[6] Annealing: Given a test tube T, all feasible double strands from single strands in T is produced

by the manipulation operation Annealing (T), which are stored in test tube T.

[7] Deanaturation: Given a test tube T, Denaturation (T) dissociates each double strand in T into two single strands.

These above mentioned manipulations are implemented with a constant number of biological steps for DNA strands [13].

Let the Input be ,T_1= { e_1 e_2, $\bar{e}_1\bar{e}_2$ }

T_2= (e_3, e_4 }

(1). Merge (T_1,T_2):

$T_1 U T_2 \Rightarrow T_1$

T_1={e_1,e_2,e_3,e_4, $\bar{e}_1\bar{e}_2$ }

(2).Copy (T_1,T_{temp}):

T_{temp}={e_1,e_2, $\bar{e}_1$,$\bar{e}_2$ }

(3)Detect(T_1):

Detect(e_1,e_2, $\bar{e}_1\bar{e}_2$ },output="yes".

Detect(), output="no".

(4). Separation(T_1,{X },T')

Separation (T_1,{e_1,$\bar{e}_2$ },T'}, T_1={ $\bar{e}_1$ }, T'={e_1,e_2, $\bar{e}_2$}

(5). Annealing (T_1):

T_1= $\left(\begin{array}{c} e_1,e_2 \\ \bar{e}_1,\bar{e}_2 \end{array}\right)$

(6) Annealing(T_1) & Denaturation(T_2):

T_1={ e_1,e_2, $\bar{e}_1$,$\bar{e}_2$ } .

(7).Annealing & Cleavage(T, $e_1 e_2$) :

T= $\left(\begin{array}{c} e_1 \\ \bar{e}_1,\bar{e} \end{array}\right)\left(\begin{array}{c} e_2 \\ 2 \end{array}\right)$

3.3 BIT REPRESENTATION WITH DNA STRANDS (STRINGS)

In this section, we give explanation of data structure for storing a set of n binary numbers using DNA strands. For additional details of binary number representations with DNA representations refer [1, 11].

Let us consider a number x such that

$x=\sum_{j=0}^{m-1} x_j*2^j$ where x_{m-1},x_{m-2},?..,x_0 are binary bits. We assume that the most significant bit x_{m-1} is a sign bit and a negative number is denoted using two's complement notation. A representation of each bit is the same as that in [1, 11], and is described below:

We first define the alphabet $\sum$ used in these representations as follows.

$\sum$ ={A_0,A_1,??,A_{n1},B_0,B_1,?.B_{m1},C_0,C_1,D_0,D, 0,#, $\bar{A}_0$, $\bar{A}_{1,,,??.}$?, $\bar{A}_{n-1}$, $\overline{B}_0$, $\overline{B}_1$,?$\overline{B}_{m-1}$, $\overline{C}_0$, $\overline{C}_1$,$\overline{D}_0$, $\overline{D}_1$, $\overline{1}$, $\overline{0}$,$\overline{\#}$ }

A_0,A_1,??A_{n-1} denote addresses of binary numbers ,and B_0,B_1,?.B_{m-1} denote bit positions in a binary number.C_0, $C_1$and D_0, D_1 are specified symbols which cut by cleavage."0" and "1" symbols are used to denote bit value and "#" is a special symbol used for separation.

Using the above defined alphabet, a value of a bit, whose address and bit position are i and j, is represented by a single strand Si, j such that

$S_{i,j}$= $D_1 A_i B_j C_0 C_1 V_{i,j} D_0$,

$V_{i,j}$ =0 if value of bit is 0, otherwise $V_{i,j}$ =1.Where $S_{i,j}$ is a memory strand, and use a set of O(mn) different memory strands to denote n binary numbers of m bits ,that is , a number x stored in address i is represented by a set of memory strands {Si,m-1,Si,m-2??.Si,0},which denote binary bits xm-1,xm-2,??x0,respectively. We assume that Vi denotes a value stored in address i, that is,

$V_i=\sum_{j=0}^{m-1} x_j*2^j$

3.4 PRIMITIVE OPERATIONS

In this paper, three operations Value assignment, logic and addition are used as primitive operations for multiplications. The ValueAssignment is a primitive operation, we express ValueAssignment such as ValueAssignment_V(T_{input}, T_{output}) executes assignments of the same value V to T_{input} ,and stores the results in T_{output} . The Logic primitive operation is expressed such as Logic(T_{input}, L,T_{output}),it as an operation which executes logic operations, which are defined by single strands in a test tube L, for pairs of memory strands in test

tube T_{input} ,and it stores the results in test tube T_{output} . The addition primitive operation expressed such as Addition (T_{input}, R,T_{output}), is an operation which executes addition, which are define by single strands in a test tube R,for pairs of memory strands in T_{input} ,and results in T_{output} .

For these three primitive operations, the following lemmas are obtained [11].

Lemma 1 [11] The ValueAssignment_V(T_{input}, T_{output}),which is for O(n) pairs of m bit binary numbers, can be executed in O(1) steps using O(1) kinds of O(mn) DNA strands.
Lemma 2 [11] The Logic (T_{input}, L,T_{output}),which is for O(n) pairs of m bit binary numbers , can be executed in O(1) steps using O(mn) kinds of different DNA strands.
Lemma 3 [11]The Addition(T_{input}, R,T_{output}) ,which is for O(n) pairs of m bit binary numbers ,can be executed in O(1) steps using O(mn) kinds of different DNA strands.

4. PROCEDURE FOR MULTIPLICATION USING DNA STRANDS

An Input and output of multiplication procedures are following three test tubes T_{input_x}, T_{input_y} and T_{output}. [Refer [1] for details]

$T_{input\_M} = \{S_{M,j} | 0 \le j \le m-1\}$

$T_{input\_Q} = \{S_{Q,j} | 0 \le j \le m-1\}$

$T_{output\_S} = \{S_{S,j} = | 0 \le j \le 2m-1\}$

In the procedure for multiplication, memory strands in T_{input_Q} and T_{input_M} denote a multiplicand and a multiplier, respectively. An output value of the multiplication is stored in memory strands in the following test tube T_{output_S}.

The DNA manipulation operations as mentioned in section ? are applied for solving multiplication of two n bit binary number. The following steps are carried out :

Step 1: Prepare an array R such that it contains the positions of all 1's in Q. This step consists of following sub steps:

(1-1) We first detect for .0 in test tube Q, and then no operation is performed if it returns "yes".
Detect (T_{input_Q}) ="yes"
// detect strands in test tube T_{input_Q};if bit is 0 then no operation performed .

(1-2) We detect position of all 1's in Q, then use logic primitive operation and separation DNA computing model for storing all 1's in a new test tube.

Detect (T_{input_Q}) ="yes"
// detect strands in test tube T_{input_Q};
Logic (T_{input_Q},L_{assign},$T_{judge\_position\_1}$);
//Logic operation for location assign in test tube T_{input_Q} and its position store in New test tube $T_{judge\_position\_1}$.
Separation(T_{input_Q},1,T_{R_1});
Annealing (T_{R_1});
//separate 1 from Input test tube Q and produce a new test tube T_{R_1} after annealing.

(1-3) Then we save all these bit positions to a test tube $T_{judge\_position\_1}$.

Separation ($T_{judge\_poition\_1}$,{1},T_{R1}}
Annealing($T_{judge\_1\_positions}$);
//Test tube $T_{judge\_1\_positions}$ used for storing the position of bits.

(1-4) Set initial Value of S=0 into T_S and A=00001011 into T_A .

ValueAssignment_0(T_{all_0},T_s)
//Set initial value of S=0
ValueAssignment_00001011(T_{tmp},T_A)
// Set Initial Value A=00001011 .

Step 2: Repeat the following steps for all values stored in R.
(2-1) Store R [i] times left shifted value of M to A and add temporarily $T_{preanswer\_A}$ Value to T_S .

for(i=0;i<=$T_{judge\_position\_1}$;i++)
{
If ($T_{judge\_position\_1}$ >=0)
{
Separation (T_{input_Q},1,T_{R_1});
Merge (T_{R_1},$T_{judge\_position\_1}$);
Annealing (T_{R_1});
Separation (T_{R_1},{$\overline{D_0D_1}$},$T_{judge\_position\_1}$);
Merge (T_{R_1},D_1);
Annealing (T_{R_1});
Cleavage (T_{R_1},D_0D_1);
Denaturation (T_{R_1});
Separation (T_{R_1},{$\overline{D_1}$},T_{shift_A});
Annealing (T_{shift_A});
Merge (T_{input_M},T_{shift_A});

Separation (T_{shift_A},{D_1},$T_{judge\_position\_1}$);
Denaturation (T_{shift_A});
Separation ($T_{shift\_A,}$ {C_0C_1},$T_{pre\_answer\_A}$);
Addition($T_{pre\_answer\_A}$, R,T_S);
//Addition performed and output stored in T_S (according to loop) S=S+A, where R is a logic which defined for additon operation.
}

Step 3: Output a value is stored in address T_S.

Separation(T_S,{A_S},T_S);
// finally output value stored in last output test tube T_S in given above separation operation, because input test tube strings removed which same as A_S.
}

In addition, we use the following test tubes:

- T_{input_Q}={$S_{Q,j}$ | 0 ≤ j ≤ m-1 }
- Tinput_M= {$S_{M,j}$| 0 ≤ j≤ m-1}
- $T_{pre\_answe\_A}$ ={$S_{A,j}$ | 0 ≤ j ≤m-1}
- Tjudge_position_1={ $\overline{B_j C_0 C_1 I D_0 D_i A_j}$ | 0 ≤ j ≤ m-1 }

5. CONCLUSIONS

In this paper, we proposed an efficient algorithm for multiplication of two n bit binary numbers. Above algorithm takes O(1) time in the best case which exists when the each bit of multiplicand is zero. However, for rest of the cases the time complexity of proposed algorithm is O(n). Similarly, the space complexity of proposed algorithm is O(n) for Average case, Worst case and Best case. In addition the most merit of this model is simple coding and its time efficiency.

6. REFERENCES

[1]. Hiroki Fukagaw, Akihiro Fujiwara; Procedures for multiplication and division in DNA computing.

[2]. L. M. Adleman; Computing with DNA. Scientific American, 279(2):54-61, 1998.

[3] V. Gupta, S. Parthasarathy, and M. J. Zaki. Arithmetic and logic operations with DNA. In Proceedings 3rd DIMACS Workshop on DNA Based Computers, pages 212-220, 1997.

[4]. H. Hug and R. Schuler. DNA-based parallel computation of simple arithmetic. In Proceedings of International Meeting on DNA Based Computers, pages 159-166, 2001.

[5]. R. J. Liption. DNA solution of hard computational problems. Science, 268:542-545, 1995.

[6] Z. F. Qiu and M. Lu. Arithmetic and logic operations for DNA computers. In Proceedings of the Second IASTED International conference on Parallel and Distributed Computing and Networks, pages 481-486,
1998.

[7] Z. F. Qiu and M. Lu. Take advantage of the computing power of DNA computers. In Proceedings of the Third Workshop on Bio-Inspired Solutions to Parallel Processing Problems, IPDPS 2000 Workshops, pages 570-577, 2000.

[8]. J. H. Reif. Parallel bimolecular computation: Models and simulations. Algorithmica, 25(2/3): 142-175, 1999.

[9] R. B. Merrifield. Solid phase peptide synthesis. I. The synthesis of a tetra peptide. Journal of the American Chemical Society, 85:2149-2154, 1963.

[10] P. Frisco. Parallel arithmetic with splicing. Romanian Journal of Information Science and Technology, 2(3):113-128, 2000.

[11]. A. Fujiwara, K. Matsumoto, and W. Chen. Addressable Procedures for logic and arithmetic operations with DNA molecules. International Journal of Foundations of Computer Science, 15(3):461-474, 2004.

[12]. F. Guarnieri, M. Fliss, and C. Bancroft. Making DNA add. Science, 273:220-223, 1996.

[13]. G. Paun, G. Rozeberg, and A. Salomaa. DNA computing. Springer-Verlag, 1998.

[14].S. Kamio, A. Takehara, and A. Fujiwara. Procedures for computing the maximum with dna strands. In Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications, volume 1, pages 351-357, 2003.