

Procedures for multiplication and division in DNA computing

Hiroki Fukagawa, Akihiro Fujiwara

Department of Computer Science and Electronics
Kyushu Institute of Technology
680-4 Kawazu, Iizuka, Fukuoka 820-8502, JAPAN
E-mail: fujiwara@cse.kyutech.ac.jp

Abstract: *In this paper, we propose two procedures for multiplication and division using DNA strands. We first show a procedure for multiplication of a pair of two binary numbers. The procedure executes multiplication for two binary numbers of m bits in $O(\log m)$ steps using $O(m^2)$ DNA strands. The procedure mainly consists of bit-shift and addition operations. We next show a procedure for division of a pair of two binary numbers. The procedure executes division for two binary numbers of m bits in $O(\log m)$ steps using $O(m^2)$ DNA strands. The procedure first computes a reciprocal of a number using Newton's method, and then, computes a quotient of the division using multiplication and subtraction. The procedure also computes a remainder of the division using multiplication and subtraction.*

Keywords: DNA computing, multiplication, division

1 Introduction

In recent works for high performance computing, computation with DNA molecules, that is, DNA computing, has considerable attention as one of non-silicon based computing. DNA molecules have two important features, which are Watson-Crick complementarity and massive parallelism. Using the features, we can solve some NP optimization problems, which usually need exponential time on a silicon based computers, in a polynomial number of steps with DNA molecules.

However, procedures for primitive operations, such as logic or arithmetic operations, are needed to apply DNA computing on a wide range of problems. A number of procedures have been proposed for the primitive operations with DNA molecules [2, 3, 4, 5, 6, 10]. Guarnieri et.al. [4] proposed an algorithm for an addition of two binary numbers of m bits. The procedure works in $O(m)$ steps using $O(m)$ different DNA strands. Hug et al. [6] proposed a model for representing and manipulating binary numbers on the DNA chip, which allows parallel execution of primitive operations. Their procedure computes an addition of two binary numbers of m bits in $O(1)$ steps using $O(m)$ different DNA molecules. Fujiwara et al. [3] proposed addressable procedures for the primitive operation. They first showed a DNA representation of n binary numbers of m bits, and they proposed procedures which compute logic operations and additions of pairs of two binary numbers. The procedures run in $O(1)$ steps using $O(mn)$ DNA strands. Kamio et al. [7] proposed a procedure which computes the maximum of n binary numbers of m bits. The procedure runs in $O(1)$ steps using $O(mn^2)$ DNA strands.

In this paper, we propose two procedures for multiplication and division with DNA strands. We first show a procedure for multiplication of a pair of two binary numbers. The procedure executes multiplication for two binary numbers of m bits in $O(\log m)$ steps using $O(m^2)$ DNA strands. The procedure mainly consists of bit-shift and addition operations. We next show a procedure for division of a pair of two binary numbers. The procedure executes division for two binary numbers of m bits in $O(\log m)$ steps using $O(m^2)$ DNA strands. The procedure first computes a reciprocal of a number using Newton's method, and then, computes a quotient of the division using multiplication and subtraction. The procedure also computes a remainder of the division using multiplication and subtraction.

2 Preliminaries

2.1 Computational model for DNA computing

A number of theoretical or practical computational models have been proposed for DNA computing [1, 5, 6, 8, 10, 11, 12]. A computational model used in this paper is the same model as [3]. We briefly introduce the model in this subsection.

A *single strand* of DNA is defined as a string of symbols over a finite alphabet Σ . We define the alphabet $\Sigma = \{\sigma_0, \sigma_1, \dots, \sigma_{m-1}, \bar{\sigma}_0, \bar{\sigma}_1, \dots, \bar{\sigma}_{m-1}\}$, where the symbols $\sigma_i, \bar{\sigma}_i$ ($0 \leq i \leq m-1$) are *complements*. Two single strands form a *double strand* if and only if the single strands are complements of each other. A double strand with $\sigma_i, \bar{\sigma}_i$ is denoted by $\begin{bmatrix} \sigma_i \\ \bar{\sigma}_i \end{bmatrix}$.

The single or double strands are stored in a *test tube*. For example, $T_1 = \{\sigma_0\sigma_1, \overline{\sigma_1\sigma_0}\}$ denotes a test tube in which two single strands $\sigma_0\sigma_1, \overline{\sigma_1\sigma_0}$ are stored.

Using the DNA strands, the following eight DNA manipulations are allowed on the computational model. Since these eight manipulations are implemented with a constant number of biological steps for DNA strands [9], we assume that the complexity of each manipulation is $O(1)$. (See [3] for details of the manipulations.)

- (1) *Merge*: Given two test tubes T_1, T_2 , $Merge(T_1, T_2)$ stores the union $T_1 \cup T_2$ in T_1 .
- (2) *Copy*: Given a test tube T_1 , $Copy(T_1, T_2)$ produces a test tube T_2 with the same contents as T_1 .
- (3) *Detect*: Given a test tube T , $Detect(T)$ outputs “yes” if T contains at least one strand, otherwise, $Detect(T)$ outputs “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) *Selection*: Given a test tube T_1 and an integer L , $Selection(T_1, L, T_2)$ removes all single strands, whose length is L , from T_1 , and produces a test tube T_2 with the removed strands. (The length of a strand is the number of symbols in the strand.)
- (6) *Cleavage*: Given a test tube T and a string of two symbols $\sigma_0\sigma_1$, $Cleavage(T, \sigma_0\sigma_1)$ cuts each double strand containing $\begin{bmatrix} \sigma_0\sigma_1 \\ \bar{\sigma}_0\bar{\sigma}_1 \end{bmatrix}$ in T into two double strands as follows.

$$\begin{bmatrix} \alpha_0\sigma_0\sigma_1\beta_0 \\ \alpha_1\bar{\sigma}_0\bar{\sigma}_1\beta_1 \end{bmatrix} \Rightarrow \begin{bmatrix} \alpha_0\sigma_0 \\ \alpha_1\bar{\sigma}_0 \end{bmatrix}, \begin{bmatrix} \sigma_1\beta_0 \\ \bar{\sigma}_1\beta_1 \end{bmatrix}$$

(We assume that *Cleavage* can only be applied to some *specified symbols* over the alphabet Σ .)

- (7) *Annealing*: Given a test tube T , $Annealing(T)$ produces all feasible double strands from single strands in T . (The produced double strands are still stored in T after *Annealing*.)
- (8) *Denaturation*: Given a test tube T , $Denaturation(T)$ dissociates each double strand in T into two single strands.

2.2 Representation of binary numbers with DNA strands

In this subsection, we explain a data structure for storing a set of n binary numbers using DNA strands. Let us consider a number x such that $x = \sum_{j=0}^{m-1} x_j * 2^j$, where $x_{m-1}, x_{m-2}, \dots, x_0$ 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 [3], and is briefly described in the following.

We first define the alphabet Σ as follows.

$$\Sigma = \{A_0, A_1, \dots, A_{n-1}, B_0, B_1, \dots, B_{m-1}, C_0, C_1, D_0, D_1, 1, 0, \#, \bar{A}_0, \bar{A}_1, \dots, \bar{A}_{n-1}, \bar{B}_0, \bar{B}_1, \dots, \bar{B}_{m-1}, \bar{C}_1, \bar{C}_2, \bar{D}_1, \bar{D}_2, \bar{1}, \bar{0}, \bar{\#}\}$$

In the above alphabet, A_0, A_1, \dots, A_{n-1} denote addresses of numbers, and B_0, B_1, \dots, B_{m-1} denote bit positions. C_0, C_1 and D_0, D_1 are specified symbols cut by *Cleavage*. Symbols “0” and “1” are used to denote values of bits, and “#” is a special symbol for *Separation*.

Using the above alphabet, a value of a bit, whose address and bit position are i and j , is represented by a single strand $S_{i,j}$ such that

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

where $V_{i,j} = 0$ if a value of the bit is 0, otherwise, $V_{i,j} = 1$. We call each $S_{i,j}$ 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 $\{S_{i,m-1}, S_{i,m-2}, \dots, S_{i,0}\}$, which denote binary bits $x_{m-1}, x_{m-2}, \dots, x_0$, respectively. We assume that V_i denotes a value stored in address i , that is,

$$V_i = \sum_{j=0}^{m-1} V_{i,j} * 2^j.$$

2.3 Input and output

An input of two procedures are following two test tubes T_{input_x} and T_{input_y} .

$$T_{input_x} = \{S_{x,j} \mid 0 \leq j \leq m-1\}, \quad T_{input_y} = \{S_{y,j} \mid 0 \leq j \leq m-1\}$$

In the procedure for multiplication, memory strands in T_{input_x} and T_{input_y} denote a multiplicand and a multiplier, respectively. Similarly, in the procedure for division, memory strands in T_{input_x} and T_{input_y} denote a dividend and a divisor, respectively.

Output test tubes are different between two procedures. In the procedure for multiplication, an output value of the multiplication is stored in memory strands in the following test tube T_{output} .

$$T_{output} = \{S_{z,j} \mid 0 \leq j \leq 2m-1\}$$

On the other hand, in the procedure for division, values of a quotient and a remainder of the division are stored in memory strands in the following test tubes $T_{quotient}$ and $T_{remainder}$, respectively.

$$T_{quotient} = \{S_{q,j} \mid 0 \leq j \leq m-1\}, \quad T_{remainder} = \{S_{r,j} \mid 0 \leq j \leq m-1\}$$

2.4 Primitive operations

In this paper, five operations *ValueAssignment*, *Logic*, *Addition*, *Subtraction* and *Minimum* are used as primitive operations. The $ValueAssignment_V(T_{input}, T_{output})$ is an operation which executes assignments of the same value V to $O(mn)$ memory strands in T_{input} , and stores the results in T_{output} . The $Logic(T_{input}, L, T_{output})$ is an operation which executes logic operations, which are defined by single strands in a test tube L , for pairs of memory strands in T_{input} , and stores the results in T_{output} . The $Addition(T_{input}, R, T_{output})$ and $Subtraction(T_{input}, R, T_{output})$ are operations which execute additions and subtractions, which are defined by single strands in a test tube R , for pairs of memory strands in T_{input} , and stores the results in T_{output} . The $Minimum(T_{input}, T_{output})$ is an operation which computes the minimum for memory strands in T_{input} , and stores the results in T_{output} .

For these five primitive operations, the following lemmas are obtained [3, 7].

Lemma 1 [3] *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.* \square

Lemma 2 [3] *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.* \square

Lemma 3 [3] *The $Addition(T_{input}, R, T_{output})$ and $Subtraction(T_{input}, R, T_{output})$, which are for $O(n)$ pairs of m bit binary numbers, can be executed in $O(1)$ steps using $O(mn)$ kinds of different DNA strands.* \square

Lemma 4 [7] *The $Minimum(T_{input}, T_{output})$, which computes the minimum of n numbers of m bits, can be executed in $O(1)$ steps using $O(mn^2)$ different DNA strands.* \square

3 Procedure for multiplication

In this section, we show a procedure for multiplication of a pair of two binary numbers. This procedure executes multiplication of two binary numbers of m bits in $O(\log m)$ steps using $O(m^2)$ DNA strands. In the following, we assume that $m = 2^p$, where p is an positive integer, and X and Y denote a multiplicand and a multiplier, respectively.

3.1 An overview of the procedure

The procedure for multiplication mainly consists of bit-shift and addition operations. A basic idea of the procedure is as follows. We first execute the following operation for each bit of Y .

- If the value of the j -th bit of Y is 1, we store the j left-shifted value of X in an address $z + j$.
- If the value of the j -th bit of Y is 0, we store the value 0 in an address $z + j$.

In both cases, a value stored in an address $z + j$ is a binary number of $2m$ bits. Then, we can obtain an output of the multiplication by adding m binary numbers in addresses $z, z + 1, \dots, z + m - 1$.

We show an example of the multiplication. Let input binary numbers be $(1011)_2$ and $(1010)_2$. Figure 1 shows execution of the multiplication. Since the values of the first and third bits of Y are 1, the shifted values of X are stored in addresses $z + 1$ and $z + 3$. On the other hand, values in addresses z and $z + 2$ are 0 because the second and fourth bits of Y are 0. The result of the multiplication is obtained by adding four binary numbers in addresses $z, z + 1, z + 2$ and $z + 3$.

				1	0	1	1	...	address x
			×	1	0	1	0	...	address y
		0	0	0	0	0	0	0	...
		0	0	0	1	0	1	1	0
		0	0	0	0	0	0	0	0
+)	0	1	0	1	1	0	0	0
		0	1	1	0	1	1	1	0

Figure 1: Multiplication of two binary numbers

To realize the above idea, we propose the following procedure.

Step 1: Prepare m binary numbers of $2m$ bits such that the k -th number denotes a k left-shifted value of X . This step consists of a manipulation *Logic*, which is described in 2.4, and some basic manipulations.

Step 2: For each j -th bit of Y ($0 \leq j \leq m - 1$), copy the j -th prepared number to a value in an address $z + j$ if a value of j -th bit is 1, otherwise, copy value 0 to a value in an address $z + j$. This step consists of the following two substeps.

(2-1) In all bits of Y , the bits, whose values are 1, are selected using a manipulation *Separation*. Then, copy the left-shifted values, which are prepared in Step 1, for the selected bits according to the bit position.

(2-2) In all bits of Y , the other bits, whose values are 0, are selected using a manipulation *Separation*. Then, copy value 0 for the selected bits.

Step 3: Repeat the following operation from $l = 0$ to $l = \log m - 1$.

For all k ($0 \leq k \leq \frac{m}{2^{l+1}-1}$), add two values in addresses $z + k$ and $z + k + \frac{m}{2^{l+1}}$ and store the result in address $z + k$, in parallel.

Step 4: Output a value stored in address z .

Procedure *Multiplication*($T_{input_x}, T_{input_y}, T_{output}$) {

Step 1:

$Copy(T_{all_0}, T_{shift_x});$
 $Separation(T_{shift_x}, \{A_z\}, T_{tmp});$
 $Merge(T_{tmp}, T_{input_x});$
 $Logic(T_{tmp}, L_{assign}, T_{tmp});$

for ($l = 0; l \leq \log m - 1; l++$) {
 $Logic(T_{shift_x}, L_{shift}, T_{shift_x});$
 $}$

Step 2:

(2-1)

$Separation(T_{input_y}, 1, T_1);$
 $Merge(T_1, T_{judge_0});$
 $Annealing(T_1);$
 $Separation(T_1, \{\overline{D_0D_1}\}, T_{trash});$
 $Merge(T_1, D_1);$
 $Annealing(T_1);$
 $Cleavage(T_1, D_0D_1);$
 $Denaturation(T_1);$
 $Separation(T_1, \{D_1\}, T_{shift});$
 $Annealing(T_{shift_x});$
 $Separation(T_{shift_x}, \{D_1\}, T_{trash});$
 $Denaturation(T_{shift_x});$
 $Separation(T_{shift_x}, \{C_0C_1\}, T_{pre_answer});$

(2-2)

$Separation(T_{input_y}, 0, T_0);$
 $Merge(T_0, T_{judge_0});$
 $Annealing(T_0);$
 $Separation(T_0, \{\overline{D_0D_1}\}, T_{trash});$
 $Merge(T_0, D_1);$
 $Annealing(T_0);$
 $Cleavage(T_0, D_0D_1);$
 $Denaturation(T_0);$
 $Separation(T_0, \{D_1\}, T_{all_0});$
 $Annealing(T_{all_0});$
 $Separation(T_{all_0}, \{D_1\}, T_{trash});$
 $Denaturation(T_{all_0});$
 $Separation(T_{all_0}, \{C_0C_1\}, T_{pre_answer});$

Step 3:

for ($l = 0; l \leq \log m - 1; l++$) {
 $Addition(T_{pre_answer}, R, T_{pre_answer});$
 $}$

Step 4: $Separation(T_{pre_answer}, \{A_z\}, T_{output});$
 $}$

Figure 2: A procedure for multiplication.

Figure 2 shows details of the procedure. (We omit explanation of the procedure due to space limitation.) In addition, the following test tubes are prepared before beginning of the procedure.

- $T_{input_x} = \{S_{x,j} \mid 0 \leq j \leq m - 1\}$, $T_{input_y} = \{S_{y,j} \mid 0 \leq j \leq m - 1\}$
- $T_{all_0} = \{S_{z+i,j} \mid 0 \leq i \leq m - 1, 0 \leq j \leq m - 1\}$
- $T_{judge_0} = \{\overline{B_jC_0C_1D_0D_1A_j} \mid 0 \leq j \leq m - 1\}$, $T_{judge_1} = \{\overline{B_jC_0C_1D_0D_1A_j} \mid 0 \leq j \leq m - 1\}$

Each step of the procedure in Figure 2 can be executed in $O(1)$ steps, and number of DNA strands required in the procedure is $O(m^2)$. Therefore, we obtain following theorem.

Theorem 1 *A procedure Multiplication executes multiplication of two binary numbers of m bits in $O(\log m)$ steps using $O(m^2)$ kinds of DNA strands.* \square

4 Procedure for division

In this section, we show a procedure *Division* for division of a pair of two binary numbers. A basic idea of the procedure is as follows. Let X and Y denote a dividend and a divisor of division, respectively.

- (1) Compute $\frac{1}{Y}$ using *the Newton method*.
- (2) Compute $X \times \frac{1}{Y}$ using the procedure for multiplication in Section 3, and obtain a quotient of the division by rounding the result of the multiplication.
- (3) Compute a remainder by subtracting a product of the quotient and Y from X .

In the above steps, we can execute (2) and (3) using a primitive operation *Subtraction* and the procedure *Multiplication* in Section 3. Thus, we explain an outline of (1) in the following.

In (1), we compute $\frac{1}{Y}$ using *the Newton method*. The newton method is a kind of root finding algorithms such that $f(x) = 0$. In this case, we define $f(x)$ as follows.

$$f(x) = Y - \frac{1}{x}$$

We now explain the Newton method briefly. Let x_0 be an initial value of a candidate for a root x . Then, it is known that repetition of the following iterative calculation converges to a root.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

In case of $f(x) = Y - \frac{1}{x}$, the above calculation becomes as follows.

$$\begin{aligned} x_{i+1} &= x_i - \frac{Y - \frac{1}{x_i}}{\frac{1}{x_i^2}} \\ &= 2x_i - Yx_i^2 \end{aligned}$$

The above calculation implies that we can compute $\frac{1}{Y}$ using multiplication, subtraction and a value of Y . In the procedure, we compute $\frac{1}{Y}$ by repeating the calculation 3 times. (Each calculation consists of *Multiplication*, *Subtraction* and some primitive operations.) In addition, we choose the initial value $x_0 = 2^{-m}$ if Y consists of m bits.

Although we omit details of the procedure *Division*, we obtain the following theorem for the procedure.

Theorem 2 *A procedure Division executes division of two binary numbers of m bits in $O(\log m)$ steps using $O(m^2)$ kinds of DNA strands.* \square

5 Conclusions

In this paper, we proposed two procedures for multiplication and division. The first procedure executes multiplication for two binary numbers of m bits in $O(\log m)$ steps using $O(m^2)$ DNA strands, and the second procedure division for the same input in $O(\log m)$ steps using $O(m^2)$ DNA strands. We are now considering a procedure for factorization using the procedure for division.

References

- [1] L. M. Adleman. Computing with DNA. *Scientific American*, 279(2):54–61, 1998.
- [2] P. Frisco. Parallel arithmetic with splicing. *Romanian Journal of Information Science and Technology*, 2(3):113–128, 2000.
- [3] A. Fujiwara, K. Matsumoto, and W. Chen. Procedures for logic and arithmetic operations with DNA molecules. *International Journal of Foundations of Computer Science*, 15(3):461–474, 2004.
- [4] F. Guarnieri, M. Fliss, and C. Bancroft. Making DNA add. *Science*, 273:220–223, 1996.
- [5] V. Gupta, S. Parthasarathy, and M. J. Zaki. Arithmetic and logic operations with DNA. In *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers*, pages 212–220, 1997.
- [6] H. Hug and R. Schuler. DNA-based parallel computation of simple arithmetic. In *Proceedings of the 7th International Meeting on DNA Based Computers*, pages 159–166, 2001.

- [7] 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.
- [8] R. J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, 1995.
- [9] G. Păun, G. Rozeberg, and A. Salomaa. *DNA computing*. Springer-Verlag, 1998.
- [10] 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.
- [11] 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.
- [12] J. H. Reif. Parallel biomolecular computation: Models and simulations. *Algorithmica*, 25(2/3):142–175, 1999.