Addressable Procedures for Logic and Arithmetic Operations with DNA Strands

A-16

Akihiro Fujiwara† Ken'ichi Matsumoto† Wei Chen‡

1 Introduction

In recent works for high performance computing, computation with DNA strands, that is, *DNA computing*, has considerable attention as one of non-silicon based computings. In this paper, we consider addressable procedures with DNA strands for primitive operations. The primitive operations are used on a silicon based computer with memory addressing, that is, each variable is stored in a memory location whose size is a constant number of bits, and an operation is executed for two memory locations indicated by addresses. If the addressing feature is used in DNA computing, we can execute different instructions for different variables stored in DNA strands.

Using a theoretical model for DNA computing, we first show representation of n binary numbers whose size is m bits, and propose a basic procedure to assign values for the representation. Since the procedure is applicable to different binary numbers independently in parallel, we can process the operation in $O(1)$ lab steps for any number of DNA strands. Next we propose a procedure for logic operations. The procedure is applicable to any boolean operation whose input and output are defined by a truth table, and works simultaneously for any pair of binary numbers in $O(1)$ lab steps using $O(mn)$ different DNA strands. Finally we propose a procedure for additions of pairs of two binary numbers. The procedure uses the methods for the Hamiltonian path problem[1] to compute carry bits and works in $O(1)$ lab steps using $O(mn)$ different DNA strands for $O(n)$ additions of two m-bit binary numbers.

2 Preliminaries

A set of DNA strands is a key component for DNA computing, like a memory module used on a silicon based computer. A *single strand* of DNA is a string of four different base nucleotides. Since any kind of single strands can be synthesized using biological methods[2], each single strand is used to represent a string over a finite alphabet Σ. A main concept used in DNA computing is Watson-Click complementation. We define the alphabet $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{m-1}, \overline{\sigma}_0, \overline{\sigma}_1, \ldots, \overline{\sigma}_{m-1}\}$, where the symbols $\sigma_i, \overline{\sigma}_i$ are *complements*. Two single strands form a *double strand* if the single strands are complements of each other. A double strand with $\sigma_i, \overline{\sigma}_i$ is denoted by $\begin{bmatrix} \sigma_i \\ \overline{\sigma}_i \end{bmatrix}$. Note that two single strands form a double strand if subsequences of the two single strands are complements. The single and double strands are stored in a *test tube*. For example, $T = \{\sigma_0\sigma_1, \overline{\sigma}_1\overline{\sigma}_0\}$ denotes a test tube T which stores two single strands $\sigma_0\sigma_1$, $\overline{\sigma}_1\overline{\sigma}_0$.

A lot of theoretical or practical computational models have been proposed for DNA computing. In this paper, we assume a theoretical computation model based on the RDNA model[4], which is an abstract mathematical model for the performance of parallel DNA computation. The model allows 8 primitive operations, which are *Merge, Copy, Detect, Separation, Selection, Cleavage, Annealing* and *Denaturation*. The 8 primitive operations are implemented with a constant number of biological operations for DNA strands[3]. In this paper, we assume the execution time of each primitive operation is *O(1) lab step*.

3 Bit representation and a basic procedure

We first describe representation of n binary numbers whose sizes are m bits. In the representation, one single strand corresponds to one bit of a binary number. We define the alphabet Σ used in the representation such that $\Sigma = \{ A_0, A_1, \ldots, A_{n-1}, B_0, B_1, \ldots, B_{m-1}, C_0, C_1, D_0, D_1, 1, 0, \sharp, \overline{A}_0, \overline{A}_1, \ldots, \overline{A}_{n-1}, \overline{B}_0, \overline{B}_1, \ldots, \overline{B}_{m-1}, \overline{C}_1, \overline{C}_2, \overline{D}_1, \overline{D}_2, \overline{1}, \overline{0}, \overline{\sharp}\}$. In the above alphabet, $A_0, A_1, \ldots, A_{n-1}$ denote addresses of binary numbers, and $B_0, B_1, \ldots, B_{m-1}$ denote bit positions in a binary number. 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.

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,

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

where V = "0" if a value of the bit is 0, otherwise V="1". We call each $S_{i,j}$ a *memory strand*, and use a set of $O(mn)$ memory strands to manipulate n binary numbers whose sizes are m bits.

We propose a basic procedure, which is *Value assignment*, for the representation. The *Value assignment* is a procedure to set the same value to all memory strands in a test tube. An input of the procedure is a test tube T_{in} which contains memory strands such that,

$$T_{in} = \{D_1 A_i B_j C_0 C_1 V_{i,j} D_0 \mid 0 \le i \le n-1, 0 \le j \le m-1\},$$

where $V_{i,j} \in \{0, 1\}$. An output is also a test tube T_{out} such that,

$$T_{out} = \{D_1 A_i B_j C_0 C_1 V' D_0 \mid 0 \le i \le n-1, 0 \le j \le m-1\},$$

where $V' \in \{0, 1\}$. The procedure consists of two steps, and a constant number of primitive operations are executed in each step. Thus, the complexity of *Value assignment* is $O(1)$ lab steps and $O(mn)$ DNA strands.

†Kyushu Institute of Technology.
‡Nanzan University.

4 Procedure for logic operations

We show a procedure which computes logic operations for pairs of two memory strands in parallel. Let us consider a logic operation whose inputs and outputs are Boolean values V_{in_1}, V_{in_2} and V_{out_1}, V_{out_2}, respectively, and values are defined by the following truth table.

V_{in_1}	V_{in_2}	V_{out_1}	V_{out_2}
0	0	α_{00}	β_{00}
0	1	α_{01}	β_{01}
1	0	α_{10}	β_{10}
1	1	α_{11}	β_{11}

Also let the following test tube T_{in} contain two memory strands whose values are $V_{i,j}$ and $V_{g,h}$ as an input.

$$T_{in} = \{D_1A_iB_jC_0C_1V_{i,j}D_0,\ D_1A_gB_hC_0C_1V_{g,h}D_0\}$$

Then, an output of the procedure, for the logic operation is a test tube T_{out} given as follows.

$$T_{out} = \begin{cases} \{D_1A_iB_jC_0C_1\alpha_{00}D_0,\ D_1A_gB_hC_0C_1\beta_{00}D_0\} \\ \qquad (\text{if } V_{i,j} = V_{g,h} = 0) \\ \{D_1A_iB_jC_0C_1\alpha_{01}D_0,\ D_1A_gB_hC_0C_1\beta_{01}D_0\} \\ \qquad (\text{if } V_{i,j} = 0, V_{g,h} = 1) \\ \{D_1A_iB_jC_0C_1\alpha_{10}D_0,\ D_1A_gB_hC_0C_1\beta_{10}D_0\} \\ \qquad (\text{if } V_{i,j} = 1, V_{g,h} = 0) \\ \{D_1A_iB_jC_0C_1\alpha_{11}D_0,\ D_1A_gB_hC_0C_1\beta_{11}D_0\} \\ \qquad (\text{if } V_{i,j} = V_{g,h} = 1) \end{cases}$$

The procedure consists of the following 3 steps.

1. Divide memory strands into two test tubes T_0 and T_1 according to outputs of the operation for a pair of two memory strands. A memory strand whose output value is 0 is stored in T_0, and the other is stored in T_1.
2. Assign values to memory strands in each of test tubes T_0, T_1.
3. Merge two test tubes.

The second and third steps are easily executed using the *Value assignment* and *Merge* operations, respectively. The first step is executed using the following *logic strands* $L_{i,j}$, which denotes a truth table of the logic operation for memory strands $S_{i,j}$, $S_{g,h}$.

$$\begin{aligned} L_{i,j,g,h} = \ & \{\overline{\alpha_{00}\sharp\ D_0S_{i,j}(0)S_{g,h}(0)D_1\beta_{00}\sharp}, \\ & \overline{\alpha_{01}\sharp\ D_0S_{i,j}(0)S_{g,h}(1)D_1\beta_{01}\sharp}, \\ & \overline{\alpha_{10}\sharp\ D_0S_{i,j}(1)S_{g,h}(0)D_1\beta_{10}\sharp}, \\ & \overline{\alpha_{11}\sharp\ D_0S_{i,j}(1)S_{g,h}(1)D_1\beta_{11}\sharp}\} \end{aligned}$$

Since the procedure consists of a constant number of primitive operations, we obtain the following theorem.

Theorem 1 *We can compute $O(n)$ logic operations for two m-bit binary numbers in $O(1)$ lab steps using $O(mn)$ different DNA strands.* □

In addition, we can compute other simple operations, such as NOT, SHIFT or COPY, with the same complexity using DNA strands because the procedure is applicable to any pair of memory strands.

5 Procedure for arithmetic operations

We consider addition of two binary numbers $a_{m-1}a_{m-2}\dots a_0$ and $b_{m-1}b_{m-2}\dots b_0$ which represent two numbers a, b such that $a = \sum_{j=0}^{m-1} a_j * 2^j$ and $b = \sum_{j=0}^{m-1} b_j * 2^j$. We assume the two numbers a, b are stored in two sets of memory strands $\{S_{i_a,m-1}, S_{i_a,m-2}, \dots, S_{i_a,0}\}$ and $\{S_{i_b,m-1}, S_{i_b,m-2}, \dots, S_{i_b,0}\}$, respectively. Then, the sum $s_{m-1}s_{m-2}\dots s_0$ of two numbers a, b is obtained using a procedure consisting of the following four steps. (Binary operators $\oplus$ and $\wedge$ are XOR and AND operations, respectively.)

1. For each j $(0 \le j \le m-1)$, compute $x_j = a_j \oplus b_j$, and $y_j = a_j \wedge b_j$
2. For each j $(0 \le j \le m-1)$, compute $p_j = x_j \wedge \neg y_j$.
3. For each j $(1 \le j \le m-1)$, set $c_j = 1$ if $y_j = 1$ or there exists k $(< j)$ such that $p_{j-1} = p_{j-2} = \dots = p_{k+1} = 1$ and $y_k = 1$, otherwise set $c_j = 0$.
4. For each j $(1 \le j \le m-1)$, set $s_j = a_j \oplus c_{j-1}$.

The first, second and fourth steps of the above procedure are easily implemented using a constant number of primitive operations and logic procedures described in Section 4. To implement the third step, we use the methods for the Hamiltonian path problem[1] to propagate carrys in the third step. Since the third step consists of a constant number of primitive operations and uses $O(mn)$ different DNA strands, we obtain the following theorem.

Theorem 2 *We can compute $O(n)$ additions of two m-bit binary numbers in $O(1)$ lab steps using $O(mn)$ different DNA strands.* □

The procedure for additions are easily modified to be applied to subtractions. Thus, we can compute subtractions with the same complexity using DNA strands.

6 Conclusions

In this paper, we proposed two procedures for logic and arithmetic operations using DNA strands. Both procedure works in $O(1)$ lab steps using $O(mn)$ different DNA strands. Our result proposed in this paper are of theoretical consequence only. There are many biological problems which should be considered to realize the procedures. However we believe that our theoretical results play an important role in future DNA computing.

References

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

[2] R. B. Merrifield. Solid phase peptide synthesis. I. The syntesis of a tetrapeptide. *Journal of the American Chemical Society*, 85:2149–2154, 1963.

[3] G. Păun, G. Rozeberg, and A. Salomaa. *DNA computing.* Springer-Verlag, 1998.

[4] J. H. Reif. Parallel biomolecular computation: Models and simulations. *Algorithmica*, 25(2/3):142–175, 1999.