Fast Parallel DNA-Based Algorithms for Molecular Computation: The Set-Partition Problem

Weng-Long Chang

Abstract—**This paper demonstrates that basic biological operations can be used to solve the set-partition problem. In order to achieve this, we propose three DNA-based algorithms, a signed parallel adder, a signed parallel subtractor and a signed parallel comparator, that formally verify our designed molecular solutions for solving the set-partition problem.**

Index Terms—**DNA-based computing, the NP-complete problems, the NP-hard problems.**

I. Introduction

In 1994 Adleman [1] succeeded in solving an instance of the Hamiltonian path problem in a test tube by handling DNA strands. From [7], it was indicated that optimal solution of every NP-complete or NP-hard problem is determined from its characteristic. DNA-based algorithms had been offered to solve many computational problems, and these contained the set-splitting problem [8], the set-cover problem and the problem of exact cover by 3-sets [9], the dominating-set [10], the independent-set problem [11], and the binary integer programming problem [12]. Potentially significant area of application for DNA algorithms is the breaking of encryption schemes [2], [3] and the constructing of DNA databases [13].

The paper is organized as follows: Section II introduces DNA models of computation proposed by Adleman and his coauthors in detail. Section III introduces the DNA program to solve the set-partition problem from solution spaces of DNA strands. Conclusions are drawn in Section IV.

II. Background

A. DNA Manipulations

Tube from [1]–[5] is a set of molecules of DNA (a multiset of finite strings over the alphabet {A, C, G, T}). Given a tube, one can perform the following operations.

i. *Extract*. Given a tube P and a short single strand of DNA, S, the operation produces two tubes $+$(P, S) and $-$(P, S), where $+$(P, S) is all of the molecules of DNA in P which contain S as a substrand and $-$(P, S) is all of the molecules of DNA in P which do not contain S.
ii. *Merge*. Given tubes P_1 and P_2, yield $\cup(\mathrm{P}_1, \mathrm{P}_2)$, where $\cup(\mathrm{P}_1, \mathrm{P}_2) = \mathrm{P}_1 \cup \mathrm{P}_2$.
iii. *Detect*. Given a tube P, if P includes at least one DNA molecule, then we have "yes." Otherwise, we have "no."
iv. *Discard*. Given a tube P, the operation will discard P.
v. *Amplify*. Given a tube P, the operation *Amplify*$(\mathrm{P}, \mathrm{P}_1, \mathrm{P}_2)$ will produce two new tubes P_1 and P_2 so that P_1 and P_2 are totally a copy of P (P_1 and P_2 are now identical) and P becomes an empty tube.
vi. *Append*. Given a tube P containing a short strand of DNA, Z, the operation will append Z onto the end of every strand in P.
vii. *Append-head*. Given a tube P containing a short strand of DNA, Z, the operation will append Z onto the head of every strand in P.
viii. *Read*. Given a tube P, the operation is used to describe a single molecule, which is contained in tube P.

Manuscript received November 24, 2004; revised October 3, 2006. This work was supported in part by the R.O.C. National Science Council under Grant 95-2221-E-151-034-.

The author is with the Department of Computer Science and Information Engineering, National Kaohsiung University of Applied Sciences, Kaohsiung 807, Taiwan, R.O.C. (e-mail: changwl@cc.kuas.edu.tw).

Digital Object Identifier 10.1109/TNB.2007.909012

III. Molecular Solutions of the Set-Partition Problem

A. The Introduction of the Set-Partition Problem

Assume that a finite set A is $\{x_n, \ldots, x_1\}$, where x_j is the jth element for $1 \leq j \leq n$. Also suppose that every element in A is a positive integer. Assume that $|A|$ is the number of elements in A and $|A|$ is equal to n. The set-partition problem is to determine whether there is a subset $A^1 \subseteq A$ such that $\sum_{x \in A^1} x = \sum_{x \in \bar{A}} x$, where $\bar{A} = \{x | x \in A \text{ and } x \notin A^1\}$. Suppose that a finite set A is $\{1, 2, 3\}$. Eight subsets, A^1, of A are $\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}$, and $\{1, 2, 3\}$. The corresponding eight subsets, $\bar{A}$, to A^1 are $\{1, 2, 3\}, \{2, 3\}, \{1, 3\}, \{1, 2\}, \{3\}, \{2\}, \{1\}$ and $\emptyset$. The sum for each pair $(A^1, \bar{A})$ is, subsequently, $(0, 6)$, $(1, 5)$, $(2, 4)$, $(3, 3)$, $(3, 3)$, $(4, 2)$, $(5, 1)$, and $(6, 0)$. So the solution of the set-partition problem for A is $\{3\}$ and $\{1, 2\}$.

B. A Pseudoalgorithm for Solving the Set-Partition Problem

From definition of the set-partition problem in Section III-A, the form of an expression, $\sum_{x \in A^1} x = \sum_{x \in \bar{A}} x$, can be transformed into another form $(3-1): \sum_{x \in A^1} x - \sum_{x \in \bar{A}} x = 0$. The following pseudoalgorithm is used to solve the set-partition problem.

```
Method 1: Solving the set-partition problem.
(1) Every computation of $\sum_{x \in A^1} x - \sum_{x \in \bar{A}} x$ for each pair $(A^1, \bar{A})$ is simultaneously performed on a molecular computer.
(2) On a molecular computer, search the answer, $\sum_{x \in A^1} x - \sum_{x \in \bar{A}} x = 0$, from the result generated by Step (1).
EndMethod
```

C. *A Library for Solving the Set-Partition Problem*

Assume that an n-bit binary number, $m_n \ldots m_1$, is applied to represent n elements in a finite set A, where the value of each bit m_j is either 1 or 0 for $1 \leq j \leq n$. From [6], for every bit m_j representing the jth element in a finite set A to $1 \leq j \leq n$, two *distinct* 15 base value sequences are designed. For the sake of convenience in our presentation, assume that m_j^1 denotes the value of m_j to be 1 and m_j^0 defines the value of m_j to be 0. Each of the 2 ^{n} different values encoding each pair $(A^1, \bar{A})$ was represented by a *library sequence* of (15^*n) bases including the concatenation of one value sequence for each bit. Library sequences are also termed *library strands* and a combinatorial pool containing library strands is termed a *library*. The following algorithm is used to construct a library to solve the set-partition problem.

```
Procedure Init($T_0$)
(1) For $j = 1$ to $n$
    (1a) Amplify($T_0, T)_1, T_2$).
    (1b) Append-head($T_1, m_j^1$).
    (1c) Append-head($T_2, m_j^0$).
    (1d) $T_0 = \cup(T_1, T_2)$.
EndFor
EndProcedure
```

Lemma 1: A library for solving the set-partition problem can be constructed from the algorithm **Init**(T_0).

Proof: Each time Step (1a) is used to amplify tube T_0 and to generate two new tubes, T_1 and T_2, which are copies of T_0, and tube T_0 becomes empty. Then, on each execution of Step (1b), it is applied to append a DNA sequence, representing the value 1 for m_j, onto the head of every strand in tube T_1. This means that the jth element in a finite set A appears in tube T_1 and it is in a subset A^1 of A but not in the corresponding subset $\bar{A}$. Each time Step (1c) is also employed to append a DNA sequence, representing the value 0 for m_j, onto the head of every strand in tube T_2. That implies that the jth element in a finite set A does not appear in tube T_2 and it is not in a subset A^1 of A but in the corresponding subset $\bar{A}$. Next, on each execution of Step (1d), it is used to pour tube T_1 and T_2 into tube T_0. This indicates that DNA strands in tube T_0 include DNA sequences of $m_j = 1$ and $m_j = 0$. After repeating execution of Steps (1a) through (1d), it finally produces tube T_0 that consists of 2^n library sequences encoding 2^n pairs $(A^1, \bar{A})$. ■

D. *Solution Space of the Value for Every Element of Each Subset for Solving the Set-Partition Problem of a Finite Set*

The value of an element x_j for $1 \leq j \leq n$ in an n-element finite set A can be represented as a *signed* binary number, $x_{j,k+1}x_{j,k} \ldots x_{j,1}$. The bit $x_{j,k+1}$ is a signed bit, the value 0, for it is used to represent positive sign and the value 1 to it is employed to represent negative sign. The bit $x_{j,k}$ is the highest order bit and the bit $x_{j,1}$ is the lowest order bit. From [6], for every bit $x_{j,a}$ to $1 \leq a \leq k+1$, two *distinct* DNA sequences are designed. For the sake of convenience in our presentation, assume that $x_{j,a}^1$ denotes the value of $x_{j,a}$ to be 1 and $x_{j,a}^0$ defines the value of $x_{j,a}$ to be 0. The following algorithm is proposed to construct library sequences encoding the value of each element in every subset from tube T_0, generated by the algorithm, **Init**(T_0).

```
Procedure Value($T_0$)
(1) For $j = 1$ to $n$
    (1a) $T_1 = +(T_0, m_j^1)$ and $T_2 = -(T_0, m_j^1)$.
    (1b) For $a = 1$ to $k$
        (1c) Append-head($T_1, x_{j,a}$).
        (1d) Append-head($T_2, x_{j,a}$).
    EndFor
    (1e) Append-head($T_1, x_{j,k+1}^0$).
    (1f) Append-head($T_2, x_{j,k+1}^1$).
    (1g) $T_0 = \cup(T_1, T_2)$.
    EndFor
EndProcedure
```

Lemma 2: Library sequences encoding the value of each element in every subset to a finite set A can be constructed from the algorithm **Value**(T_0).

Proof: Refer to **Lemma 1**. ■

E. *A Library Sequence of an Initial Value to Computation of the Sum for Elements in Each Subset in a Finite Set*

From definition of the set-partition problem denoted in Sections III-A and III-B, it is indicated that adder and subtractor of n times are used to perform computation of the sum for elements in each subset in an n-element finite set A. Assume that Y is used to represent the sum of elements in each subset in an n-element finite set A. Also suppose that the length of Y is $(k+1)$ bits and Y is represented as a $(k+1)$-bit binary number, $y_{f,k+1}, y_{f,k}, \ldots y_{f,1}$, where the value of each bit $y_{f,g}$ is either 1 or 0 for $1 \leq f \leq (n+1)$ and $1 \leq g \leq k+1$. The bit $y_{f,k+1}$ is a signed bit, the value 0, for it is used to represent positive sign and the value 1 to it is employed to represent negative sign. The bits $y_{f,k}$ and $y_{f,1}$ are employed to represent the most significant bit and the least significant bit for Y, respectively. If updating of the f th time for Y is finished through an adder, then two binary numbers $y_{f,k+1}, y_{f,k}, \ldots, y_{f,1}$ and $y_{f+1,k+1}, y_{f+1,k}, \ldots, y_{f+1,1}$ are used to represent the augend and the sum of the f th updating, respectively. If updating of the fth time for Y is finished through a subtractor, then two binary numbers $y_{f,k+1}, y_{f,k}, \ldots, y_{f,1}$ and $y_{f+1,k+1}, y_{f+1,k}, \ldots, y_{f+1,1}$ are applied to represent the minuend and the difference of the f th updating, respectively. From [6], for every bit $y_{f,g}$, two *distinct* 15 base value sequences are designed. For the sake of convenience in our presentation, assume that $y_{f,g}^1$ denotes the value of $y_{f,g}$ to be 1 and $y_{f,g}^0$ defines the value of $y_{f,g}$ to be 0. The following algorithm is used to construct a library sequence to encode an initial value to computation of the sum for elements in every pair $(A^1, \bar{A})$ to a finite set A.

```
Procedure InitialValue($T_0$)
(1) For $g = 1$ to $k + 1$
    (1a) Append-head($T_0, y^0_{1,g}$).
    EndFor
    EndProcedure
```

Lemma 3: Library strands for initial values to computation of the sum for elements in every pair $(A^1, \bar{A})$ to a finite set A can be constructed from the algorithm, **InitialValue**(T_0). **Proof**: Similar to **Lemma 1**. ∎

F. The Construction of a Parallel One-Bit Comparator

The following algorithm, **OneBitComparator**$(T_{0,1}, T_{1,0}, T^{>=}_{0,1}, T^{<}_{0,1}, T^{>=}_{1,0}, T^{<}_{1,0}, f, g, j, a)$, is presented to finish the function of a one-bit parallel comparator.

```
Procedure OneBitComparator($T_{0,1}, T_{1,0}, T^{>=}_{0,1}, T^{<}_{0,1}, T^{>=}_{1,0}, T^{<}_{1,0}, f, g, j, a$)
(1) $T^{ON}_1 = +(T_{0,1}, y^1_{f,g})$ and $T^{OFF}_1 = -(T_{0,1}, y^1_{f,g})$.
(2) $T^{ON}_2 = +(T^{ON}_1, x^1_{j,a})$ and $T^{OFF}_2 = -(T^{ON}_1, x^1_{j,a})$.
(3) $T^{ON}_3 = +(T^{OFF}_1, x^1_{j,a})$ and $T^{OFF}_3 = -(T^{OFF}_1, x^1_{j,a})$.
(4) $T^{ON}_4 = +(T_{1,0}, y^1_{f,g})$ and $T^{OFF}_4 = -(T_{1,0}, y^1_{f,g})$.
(5) $T^{ON}_5 = +(T^{ON}_4, x^1_{j,a})$ and $T^{OFF}_5 = -(T^{ON}_4, x^1_{j,a})$.
(6) $T^{ON}_6 = +(T^{OFF}_4, x^1_{j,a})$ and $T^{OFF}_6 = -(T^{OFF}_4, x^1_{j,a})$.
(7) $T_{0,1} = \cup(T_{0,1}, T^{ON}_2, T^{OFF}_3)$
(8) $T_{1,0} = \cup(T_{1,0}, T^{ON}_5, T^{OFF}_6)$.
(9) $T^{>=}_{0,1} = \cup(T^{>=}_{0,1}, T^{OFF}_2)$.
(10) $T^{<}_{0,1} = \cup(T^{<}_{0,1}, T^{ON}_3)$.
(11) $T^{>=}_{1,0} = \cup(T^{>=}_{1,0}, T^{OFF}_5)$.
(12) $T^{<}_{1,0} = \cup(T^{<}_{1,0}, T^{ON}_6)$.
EndProcedure
```

Lemma 4: The algorithm **OneBitComparator**$(T_{0,1}, T_{1,0}, T^{>=}_{0,1}, T^{<}_{0,1}, T^{>=}_{1,0}, T^{<}_{1,0}, f, g, j, a)$ can be applied to finish the function of a one-bit parallel comparator.

Proof: The execution for Steps (1) through (3) employs the *extract* operations to form six test tubes. Tube T^{ON}_1 includes library sequences that have $y_{f,g} = 1$, tube T^{OFF}_1 consists of library strands that have $y_{f,g} = 0$, tube T^{ON}_2 includes library sequences that have $y_{f,g} = 1$ and $x_{j,a} = 1$, tube T^{OFF}_2 consists of library strands that have $y_{f,g} = 1$ and $x_{j,a} = 0$, tube T^{ON}_3 includes library sequences that have $y_{f,g} = 0$ and $x_{j,a} = 1$ and tube T^{OFF}_3 consists of library strands that have $y_{f,g} = 0$ and $x_{j,a} = 0$. Next, the execution for Steps (4) through (6) applies also the *extract* operations to form six test tubes. Tube T^{ON}_4 includes library sequences that have $y_{f,g} = 1$, tube T^{OFF}_4 consists of library strands that have $y_{f,g} = 0$, tube T^{ON}_5 includes library sequences that have $y_{f,g} = 1$ and $x_{j,a} = 1$, tube T^{OFF}_5 consists of library strands that have $y_{f,g} = 1$ and $x_{j,a} = 0$, tube T^{ON}_6 includes library sequences that have $y_{f,g} = 0$ and $x_{j,a} = 1$, and tube T^{OFF}_6 consists of library strands that have $y_{f,g} = 0$ and $x_{j,a} = 0$. The execution to Steps (7) through (12) uses the *merge* operations to pour tubes T^{ON}_2 and T^{OFF}_3 into tube $T_{0,1}$, to pour tubes T^{ON}_5 and T^{OFF}_6 into tube $T_{1,0}$, to pour tube T^{OFF}_2 into tube $T^{>=}_{0,1}$, to pour tube T^{ON}_3 into tube $T^{<}_{0,1}$, to pour tube T^{OFF}_5 into tube $T^{>=}_{1,0}$ and to pour tube T^{ON}_6 into tube $T^{<}_{1,0}$. ∎

From the algorithm **OneBitComparator**$(T_{0,1}, T_{1,0}, T^{>=}_{0,1}, T^{<}_{0,1}, T^{>=}_{1,0}, T^{<}_{1,0}, f, g, j, a)$, it takes six *extract* operations, six *merge* operations, and 18 test tubes to perform the function of a one-bit parallel comparator.

G. The Construction of a Signed Parallel Comparator

The following algorithm, **ParallelComparator**$(T_0, T_{0,0}, T_{1,1}, T^{>=}_{0,1}, T^{<}_{0,1}, T^{>=}_{1,0}, T^{<}_{1,0}, f, j)$ is proposed to perform the function of a $(k + 1)$-bit signed parallel comparator.

```
Procedure ParallelComparator($T_0, T_{0,0}, T_{1,1}, T^{>=}_{0,1}, T^{<}_{0,1}, T^{>=}_{1,0}, T^{<}_{1,0}, f, j$)
(1) $T^{ON}_7 = +(T_0, y^1_{f,k+1})$ and $T^{OFF}_7 = -(T_0, y^1_{f,k+1})$.
(2) $T_{1,1} = +(T^{ON}_7, x^1_{j,k+1})$ and $T_{1,0} = -(T^{ON}_7, x^1_{j,k+1})$.
(3) $T_{0,1} = +(T^{OFF}_7, x^1_{j,k+1})$ and $T_{0,0} = -(T^{OFF}_7, x^1_{j,k+1})$.
(4) For $g = k$ to 1
(4a) OneBitComparator($T_{0,1}, T_{1,0}, T^{>=}_{0,1}, T^{<}_{0,1}, T^{>=}_{1,0}, T^{<}_{1,0}, f, g, j, g$).
(4b) If ((Detect($T_{0,1}$) = "no") and (Detect($T_{1,0}$) = "no")) then
        (4c) Terminate the execution of the loop.
    EndIf
EndFor
(5) $T^{>=}_{0,1} = \cup(T^{>=}_{0,1}, T_{0,1})$.
(6) $T^{>=}_{1,0} = \cup(T^{>=}_{1,0}, T_{1,0})$.
EndProcedure
```

Lemma 5: The algorithm **ParallelComparator**$(T_0, T_{0,0}, T_{1,1}, T^{>=}_{0,1}, T^{<}_{0,1}, T^{>=}_{1,0}, T^{<}_{1,0}, f, j)$ can be used to finish the function of a $(k + 1)$-bit signed parallel comparator.

Proof: The execution for Steps (1) through (3) employs the *extract* operations to form six test tubes. Tube T^{ON}_7 includes library sequences that have $y_{f,k+1} = 1$, tube T^{OFF}_7 consists of library strands that have $y_{f,k+1} = 0$, tube $T_{1,1}$ includes library sequences that have $y_{f,k+1} = 1$ and $x_{j,k+1} = 1$, tube $T_{1,0}$ consists of library strands that have $y_{f,k+1} = 1$ and $x_{j,k+1} = 0$, tube $T_{0,1}$ includes library sequences that have $y_{f,k+1} = 0$ and $x_{j,k+1} = 1$, and tube $T_{0,0}$ consists of library strands that have $y_{f,k+1} = 0$ and $x_{j,k+1}0$. The only loop is used to implement the function of a $(k + 1)$-bit signed parallel comparator. The

TABLE I
TRUTH TABLE OF A ONE-BIT ADDER

Augend bit	Addend bit	Previous carry bit	Sum bit	Carry bit
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

first execution of Step (4a) calls the algorithm **OneBitComparator**$(T_{0,1}, T_{1,0}, T_{0,1}^{>=}, T_{0,1}^{<}, T_{1,0}^{>=}, T_{1,0}^{<}, f, g, j, g)$ to finish the comparative result of a bit. On the first execution of Step (4b), it uses the *detect* operations to check whether there is any DNA sequence in tubes $T_{0,1}$ and $T_{1,0}$. If the two *detect* operations both return a "no," then the execution of Step (1c) will terminate the execution of the loop. Otherwise, Repeat execution of Steps (4a) through (4b) until every bit is processed. Finally, the execution for Steps (5) through (6) uses the *merge* operations to pour tube $T_{0,1}$ into tube $T_{0,1}^{>=}$ and to pour tube $T_{1,0}$ into tube $T_{1,0}^{>=}$. ■

H. The Construction of a Parallel One-Bit Adder

A one-bit adder is to perform the arithmetic sum of three input bits. It includes three inputs and two outputs. Two of the input bits are used to represent augend and addend bits to be added, respectively. The third input represents the carry from the previous lower significant position. The first output gives the value of the sum for augend and addend bits to be added. The second output gives the value of the carry to augend and addend bits to be added. The truth table of a one-bit adder is shown in Table I.

Suppose that two one-bit binary numbers denoted in Section III-E, $y_{f,g}$ and $y_{f+1,g}$, represent the first input of a one-bit adder for $1 \leq f \leq (n+1)$ and $1 \leq g \leq k+1$, and the first output of a one-bit adder, respectively, a one-bit binary number denoted in Section III-D, $x_{j,a}$, represents the second input of a one-bit adder for $1 \leq j \leq n$ and $1 \leq a \leq k+1$, and two one-bit binary numbers, $z_{f,g}$ and $z_{f,g-1}$, represent the second output and the third input of a one-bit adder, respectively. Two *distinct* DNA sequences are designed to encode the value "0" or "1" for every bit $z_{f,g-1}$ and $z_{f,g}$ to $1 \leq f \leq (n+1)$ and $1 \leq g \leq k+1$. For the sake of convenience in our presentation, assume that $z_{f,g}^1$ contains the value of $z_{f,g}$ to be 1 and $z_{f,g}^0$ contains the value of $z_{f,g}$ to be 0. Also suppose that $y_{f+1,g}^1$ denotes the value of $y_{f+1,g}$ to be 1 and $y_{f+1,g}^0$ defines the value of $y_{f+1,g}$ to be 0. Similarly, assume that $z_{f,g-1}^1$ contains the value of $z_{f,g-1}$ to be 1 and $z_{f,g-1}^0$ contains the value of $z_{f,g-1}$ to be 0. The following algorithm is proposed to perform the function of a parallel one-bit adder.

```
Procedure ParallelOneBitAdder($T_{20}, f, g, j, a$)
(1) $T_1 = +(T_{20}, y_{f,g}^1)$ and $T_2 = -(T_{20}, y_{f,g}^1)$.
(2) $T_3 = +(T_1, x_{j,a}^1)$ and $T_4 = -(T_1, x_{j,a}^1)$.
(3) $T_5 = +(T_2, x_{j,a}^1)$ and $T_6 = -(T_2, x_{j,a}^1)$.
(4) $T_7 = +(T_3, z_{f,g-1}^1)$ and $T_8 = -(T_3, z_{f,g-1}^1)$.
(5) $T_9 = +(T_4, z_{f,g-1}^1)$ and $T_{10} = -(T_4, z_{f,g-1}^1)$.
(6) $T_{11} = +(T_5, z_{f,g-1}^1)$ and $T_{12} = -(T_5, z_{f,g-1}^1)$.
(7) $T_{13} = +(T_6, z_{f,g-1}^1)$ and $T_{14} = -(T_6, z_{f,g-1}^1)$.
(8) Append-head($T_7, y_{f+1,g}^1$) and Append-head($T_7, z_{f,g}^1$).
(9) Append-head($T_8, y_{f+1,g}^0$) and Append-head($T_8, z_{f,g}^1$).
(10) Append-head($T_9, y_{f+1,g}^0$) and Append-head($T_9, z_{f,g}^1$).
(11) Append-head($T_{10}, y_{f+1,g}^1$) and
     Append-head($T_{10}, z_{f,g}^0$).
(12) Append-head($T_{11}, y_{f+1,g}^0$) and
     Append-head($T_{11}, z_{f,g}^1$).
(13) Append-head($T_{12}, y_{f+1,g}^1$) and
     Append-head($T_{12}, z_{f,g}^0$).
(14) Append-head($T_{13}, y_{f+1,g}^1$) and
     Append-head($T_{13}, z_{f,g}^0$).
(15) Append-head($T_{14}, y_{f+1,g}^0$) and
     Append-head($T_{14}, z_{f,g}^0$).
(16) $T_{20} = \cup(T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14})$.
EndProcedure
```

Lemma 6: The algorithm **ParallelOneBitAdder**(T_{20}, f, g, j, a) can be applied to finish the function of a parallel one-bit adder.

Proof: Refer to **Lemma 1** through **Lemma 5**. ■

I. The Construction of a Signed Binary Parallel Adder

The one-bit adder introduced in Section III-H figures out the sum and the carry of two input bits and a previous carry. Two signed $(k)+1$)-bit binary numbers each can be added by means of this one-bit adder. A signed binary parallel adder is to finish the arithmetic sum for two signed $(k)+1$)-bit binary numbers. The following algorithm is proposed to finish the function of a signed binary parallel adder.

```
Procedure BinaryParallelAdder($T_{0,0}, T_{1,1}, f$)
(0) If (Detect($T_{0,0}$) = "yes") then
(1) Append-head($T_{0,0}$ , $z_{f,0}^0$).
(2) $T_{20} = \cup(T_{0,0}, T_{20})$.
(3) For $g = 1$ to $k$
    (3a) ParallelOneBitAdder($T_{20}, f, g, f, g$).
```

```
        EndFor
    (4) $T_{0,0} = \cup(T_{0,0}, T_{20})$.
    (5) Append-head($T_{0,0}, y^0_{f+1,k+1}$).
    EndIf
    (5a) If (Detect($T_{1,1}$) = "yes") then
    (6) Append-head($T_{1,1}, z^0_{f,0}$).
    (7) $T_{20} = \cup(T_{1,1}, T_{20})$.
    (8) For $g = 1$ to $k$
        (8a) ParallelOneBitAdder($T_{20}, f, g, f, g$).
        EndFor
    (9) $T_{1,1} = \cup(T_{1,1}, T_{20})$.
    (10) Append-head($T_{1,1}, y^1_{f+1,k+1}$).
    EndIf
    EndProcedure
```

Lemma 7: The algorithm **BinaryParallelAdder**$(T_{0,0}, T_{1,1}, f)$ can be applied to finish the function of a signed binary parallel adder.

Proof: Refer to **Lemma 1** through **Lemma 5**. ■

J. The Construction of a Parallel One-Bit Subtractor for the Absolute Value of the First Operand Greater Than or Equal to the Absolute Value of the Second Operand

A one-bit subtractor is to finish the arithmetic subtraction of three input bits. It consists of three inputs and two outputs. Two of the input bits represent minuend and subtrahend bits to be subtracted. The third input represents the borrow bit from the previous lower significant position. The first output gives the value of the difference for minuend and subtrahend bits to be subtracted. The second output gives the value of the borrow bit to minuend and subtrahend bits to be subtracted. The truth table of a one-bit subtractor is shown in Table II.

Suppose that the two one-bit binary numbers $y_{f,g}$ and $y_{f+1,g}$ denoted in Subsection E in Section III, represent the first input and the first output of a one-bit subtractor for $1 \le f \le (n+1)$ and $1 \le g \le k+1$. Also suppose a one-bit binary number $x_{j,a}$ denoted in Section III-D, represents the second input of a one-bit subtractor for $1 \le j \le n$ and $1 \le a \le k+1$, and two one-bit binary numbers $z_{f,g}$ and $z_{f,g-1}$ denoted in Subsection H in Section III represent the second output and the third input of a one-bit subtractor. The following algorithm is proposed to finish the function of a parallel one-bit subtractor.

```
    Procedure ParallelOneBitSubtractorGE($T_{30}, f, g, j, a$)
    (1) $T_1 = +(T_{30}, y^1_{f,g})$ and $T_2 = -(T_{30}, y^1_{f,g})$.
    (2) $T_3 = +(T_1, x^1_{j,a})$ and $T_4 = -(T_1, x^1_{j,a})$.
    (3) $T_5 = +(T_2, x^1_{j,a})$ and $T_6 = -(T_2, x^1_{j,a})$.
    (4) $T_7 = +(T_3, z^1_{f,g-1})$ and $T_8 = -(T_3, z^1_{f,g-1})$.
    (5) $T_9 = +(T_4, z^1_{f,g-1})$ and $T_{10} = -(T_4, z^1_{f,g-1})$.
    (6) $T_{11} = +(T_5, z^1_{f,g-1})$ and $T_{12} = -(T_5, z^1_{f,g-1})$.
    (7) $T_{13} = +(T_6, z^1_{f,g-1})$ and $T_{14} = -(T_6, z^1_{f,g-1})$.
    (8) Append-head($T_7, y^1_{f+1,g}$) and Append-head($T_7, z^1_{f,g}$).
    (9) Append-head($T_8, y^0_{f+1,g}$) and Append-head($T_8, z^o_{f,g}$).
    (10) Append-head($T_9, y^0_{f+1,g}$) and Append-head($T_9, z^0_{f,g}$).
    (11) Append-head($T_{10}, y^1_{f+1,g}$) and
    Append-head($T_{10}, z^0_{f,g}$).
    (12) Append-head($T_{11}, y^0_{f+1,g}$) and
    Append-head($T_{11}, z^1_{f,g}$).
    (13) Append-head($T_{12}, y^1_{f+1,g}$) and
    Append-head($T_{12}, z^1_{f,g}$).
    (14) Append-head($T_{13}, y^1_{f+1,g}$) and
    Append-head($T_{13}, z^1_{f,g}$).
    (15) Append-head($T_{14}, y^0_{f+1,g}$) and
    Append-head($T_{14}, z^0_{f,g}$).
    (16) $T_{30} = \cup(T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14})$.
    EndProcedure
```

TABLE II
TRUTH TABLE OF A ONE-BIT SUBTRACTOR

Minuend bit	Subtrahend bit	Previous borrow bit	Difference bit	Borrow bit
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Lemma 8: The algorithm **ParallelOneBitSubtractor-GE**(T_{30}, f, g, j, a) can be applied to finish the function of a parallel one-bit subtractor for that the absolute value of the first operand is greater than or equal to the absolute value of the second operand.

Proof: Refer to **Lemma 1** through **Lemma 5**. ■

K. The Construction of a Signed Binary Parallel Subtractor for the Absolute Value of the First Operand Greater Than or Equal to the Absolute Value of the Second Operand

The one-bit subtractor introduced in Section III-J figures out the difference bit and the borrow bit for two input bits and a previous borrow. Two signed $(k+1)$- bit binary numbers can finish subtractions of k times by means of this one-bit subtractor. A signed binary parallel subtractor is to finish arithmetic subtraction for two signed $(k+1)$-bit binary numbers. The following algorithm is proposed to finish the function of a signed binary parallel subtractor for that the absolute value of the first operand is greater than or equal to the absolute value of the second operand.

```
Procedure BinaryParallelSubtractorGE($T_{0,1}^{>=}, T_{1,0}^{>=}, f$)
(1) If (Detect($T_{0,1}^{>=}$) = "yes") then
(2) Append-head($T_{0,1}^{>=}, z_{f,0}^{0}$).
(3) $T_{30} = \cup(T_{0,1}^{>=}, T_{30})$.
(4) For $g = 1$ to $k$
        (4a) ParallelOneBitSubtractorGE($T_{30}, f, g, f, g$).
    EndFor
(5) $T_{0,1}^{>=} = \cup(T_{0,1}^{>=}, T_{30})$.
            (6) Append-head($T_{0,1}^{>=}, y_{f+1,k+1}^{0}$).
EndIf
(7) If (Detect($T_{1,0}^{>=}$) = "yes") then
(8) Append-head($T_{1,0}^{>=}, z_{f,0}^{0}$).
(9) $T_{30} = \cup(T_{1,0}^{>=}, T_{30})$.
(10) For $g = 1$ to $k$
        (10a) ParallelOneBitSubtractorGE($T_{30}, f, g, f, g$).
    EndFor
(11) $T_{1,0}^{>=} = \cup(T_{1,0}^{>=}, T_{30})$.
(12) Append-head($T_{1,0}^{>=}, y_{f+1,k+1}^{1}$).
EndIf
EndProcedure
```

Lemma 9: The algorithm **BinaryParallelSubtractor-GE**$(T_{0,1}^{>=}, T_{1,0}^{>=}, f)$ can be applied to finish the function of a signed binary parallel subtractor for that the absolute value of the first operand is greater than or equal to the absolute value of the second operand.

Proof: Refer to **Lemma 1** through **Lemma 5**. ■

L. The Construction of a Parallel One-Bit Subtractor for the Absolute Value of the First Operand Less Than the Absolute Value of the Second Operand

Because the absolute value of the first operand is less than the absolute value of the second operand, suppose that two one-bit binary numbers $y_{f,g}$ and $y_{f+1,g}$ denoted in Section III-E are used to represent the *second* input and the first output of a one-bit subtractor for $1 \le f \le (n+1)$ and $1 \le g \le k+1$, a one-bit binary number $x_{j,a}$ denoted in Section III-D is applied to represent the *first* input of a one-bit subtractor for $1 \le j \le n$ and $1 \le a \le k+1$ and two one-bit binary numbers $z_{f,g}$ and $z_{f,g-1}$ denoted in Section III-H are employed to represent the second output and the third input of a one-bit subtractor. The following algorithm is proposed to finish the function of a parallel one-bit subtractor for that the absolute value of the first operand is less than the absolute value of the second operand.

```
Procedure ParallelOneBitSubtractorLT($T_{40}, f, g, j, a$)
(1) $T_1 = +(T_{40}, x_{j,a}^1)$ and $T_2 = -(T_{40}, x_{j,a}^1)$.
(2) $T_3 = +(T_1, y_{f,g}^1)$ and $T_4 = -(T_1, y_{f,g}^1)$.
(3) $T_5 = +(T_2, y_{f,g}^1)$ and $T_6 = -(T_2, y_{f,g}^1)$.
(4) $T_7 = +(T_3, z_{f,g-1}^1)$ and $T_8 = -(T_3, z_{f,g-1}^1)$.
(5) $T_9 = +(T_4, z_{f,g-1}^1)$ and $T_{10} = -(T_4, z_{f,g-1}^1)$.
(6) $T_{11} = +(T_5, z_{f,g-1}^1)$ and $T_{12} = -(T_5, z_{f,g-1}^1)$.
(7) $T_{13} = +(T_6, z_{f,g-1}^1)$ and $T_{14} = -(T_6, z_{f,g-1}^1)$.
(8) Append-head($T_7, y_{f+1,g}^1$) and Append-head($T_7, z_{f,g}^1$).
(9) Append-head($T_8, y_{f+1,g}^0$) and Append-head($T_8, z_{f,g}^o$).
(10) Append-head($T_9, y_{f+1,g}^0$) and Append-head($T_9, z_{f,g}^0$).
(11) Append-head($T_{10}, y_{f+1,g}^1$) and
Append-head($T_{10}, z_{f,g}^0$).
(12) Append-head($T_{11}, y_{f+1,g}^0$) and
Append-head($T_{11}, z_{f,g}^1$).
(13) Append-head($T_{12}, y_{f+1,g}^1$) and
Append-head($T_{12}, z_{f,g}^1$).
(14) Append-head($T_{13}, y_{f+1,g}^1$) and
Append-head($T_{13}, z_{f,g}^1$).
(15) Append-head($T_{14}, y_{f+1,g}^0$) and
Append-head($T_{14}, z_{f,g}^0$).
(16) $T_{40} = \cup(T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14})$.
EndProcedure
```

Lemma 10: The algorithm **ParallelOneBitSubtractor-LT**(T_{40}, f, g, j, a) can be applied to finish the function of a parallel one-bit subtractor for that the absolute value of the first operand is less than the absolute value of the second operand.

Proof: Refer to **Lemma 1** through **Lemma 5**. ■

M. The Construction of a Binary Parallel Subtractor for the Absolute Value of the First Operand Less Than the Absolute Value of the Second Operand

The following algorithm is proposed to finish the function of a binary parallel subtractor for that the absolute value of the first operand is less than the absolute value of the second operand.

```
Procedure BinaryParallelSubtractorLT($T_{0,1}^{<}, T_{1,0}^{<}, f$)
(1) If (Detect($T_{0,1}^{<}$) = "yes") then
(2) Append-head($T_{0,1}^{<}, z_{f,0}^{0}$).
(3) $T_{40} = \cup(T_{0,1}^{<}, T_{40})$.
```

```
(4) For $g = 1$ to $k$
    (4a) ParallelOneBitSubtractorLT$(T_{40}, f, g, f, g)$.
  EndFor
(5) $T_{0,1}^{<} = \cup(T_{0,1}^{<}, T_{40})$.
(6) Append-head$(T_{0,1}^{<}, y_{f+1,k+1}^{1})$.
(7) If (Detect$(T_{1,0}^{<})$ = "yes") then
(8) Append-head$(T_{1,0}^{<}, z_{f,0}^{0})$.
(9) $T_{40} = \cup(T_{1,0}^{<}, T_{40})$.
(10) For $g = 1$ to $k$
    (10a) ParallelOneBitSubtractorLT$(T_{40}, f, g, f, g)$.
  EndFor
(11) $T_{1,0}^{<} = \cup(T_{1,0}^{<}, T_{40})$.
(12) Append-head$(T_{1,0}^{<}, y_{f+1,k+1}^{0})$.
EndIf
EndProcedure
```

Lemma 11: The algorithm **BinaryParallelSubtractorLT**$(T_{0,1}^{<}, T_{1,0}^{<}, f)$ can be applied to finish the function of a binary parallel subtractor for that the absolute value of the first operand is less than the absolute value of the second operand.

Proof: Refer to **Lemma 1** through **Lemma 5**. ■

N. The Algorithm for Solving the Set-Partition Problem

The following DNA algorithm is applied to solve the set-partition problem.

Algorithm 1: Solve the set-partition problem

```
(1) Init$(T_0)$.
(2) Value$(T_0)$.
(3) InitialValue$(T_0)$.
(4) For $f = 1$ to $n$
    (4a) ParallelComparator$(T_0, T_{0,0}, T_{1,1}, T_{0,1}^{>=}, T_{0,1}^{<}, T_{1,0}^{>=}, T_{1,0}^{<}, f, f)$.
    (4b) BinaryParallelAdder$(T_{0,0}, T_{1,1}, f)$.
    (4c) BinaryParallelSubtractorGE$(T_{0,1}^{>=}, T_{1,0}^{>=}, f)$.
    (4d) BinaryParallelSubtractorLT$(T_{0,1}^{<}, T_{1,0}^{<}, f)$.
    (4e) $T_0 = \cup(T_{0,0}, T_{1,1}, T_{0,1}^{>=}, T_{0,1}^{<}, T_{1,0}^{>=}, T_{1,0}^{<})$.
EndFor
(5)For $g = 1$ to $k$
    (5a) $T_0 = +(T_0, y_{n+1,g}^{0})$ and $T_{50} = -(T_0, y_{n+1,g}^{0})$.
    (5b) Discard$(T_{50})$.
EndFor
(6) If (Detect$(T_0)$ = "yes") then
    (6a) Read$(T_0)$.
EndIf
EndAlgorithm
```

Theorem 1: From those steps in Algorithm 1, the set-partition problem to a finite n-element set A can be solved.

Proof: On the execution of Step (1), it calls **Init**(T_0) to construct library sequences for 2^n pairs of subsets, $(A^1, \bar{A})$, for the set-partition problem to a finite n-element set, A. This means that tube T_0 includes library sequences encoding 2 ^{n} possible solutions for the set-partition problem to a finite n-element set, A. Next, the execution of Step (2) calls **Value**(T_0) to perform to encode the value of every element in each pair $(A^1, \bar{A})$. This implies that the value of every element in each pair $(A^1, \bar{A})$ is encoded by library sequences in tube T_0. On the execution of Step (3), it calls **InitialValue**(T_0) to encode an initial value of the sum for every element in each pair $(A^1, \bar{A})$. This indicates that tube T_0 contains library sequences encoding an initial value of the sum for every element in each pair $(A^1, \bar{A})$.

Next, Step (4) is a single loop and is mainly used to perform computation of the sum for every element in each pair $(A^1, \bar{A})$. On each execution of Step (4a), it calls **ParallelComparator**$(T_0, T_{0,0}, T_{1,1}, T_{0,1}^{>=}, T_{0,1}^{<}, T_{1,0}^{>=}, T_{1,0}^{<}, f, f)$ to classify library sequences in tube T_0 into six classes from their values. Two operands of an addition encoded by library sequences in tube $T_{0,0}$ are positive integers and two operands of an addition encoded by library sequences in tube $T_{1,1}$ are negative integers. Two operands of an addition encoded by library sequences in tube $T_{0,1}^{>=}$ are, subsequently, a positive integer and a negative integer and the absolute value of the first operand is greater than or equal to the absolute value of the second operand. Two operands of an addition encoded by library sequences in tube $T_{0,1}^{<}$ are, subsequently, a positive integer and a negative integer and the absolute value of the first operand is less than the absolute value of the second operand. Two operands of an addition encoded by library sequences in tube $T_{1,0}^{>=}$ are, subsequently, a negative integer and a positive integer and the absolute value of the first operand is greater than or equal to the absolute value of the second operand. Two operands of an addition encoded by library sequences in tube $T_{1,0}^{<}$ are, subsequently, a negative integer and a positive integer and the absolute value of the first operand is less than the absolute value of the second operand. Next, on each execution of Step (4b), it calls **BinaryParallelAdder**$(T_{0,0}, T_{1,1}, f)$ to perform computation of addition for library sequences in tube $T_{0,0}$ and in tube $T_{1,1}$. Each execution for Step (4c) calls **BinaryParallelSubtractorGE**$(T_{0,1}^{>=}, T_{1,0}^{>=}, f)$ to perform computation of subtraction for library sequences in tube $T_{0,1}^{>=}$ and in tube $T_{1,0}^{>=}$. Next, on each execution of Step (4d), it calls **BinaryParallelSubtractorLT**$(T_{0,1}^{<}, T_{1,0}^{<}, f)$ to perform computation of subtraction for library sequences in tube $T_{0,1}^{<}$ and in tube $T_{1,0}^{<}$. Next, each execution of Step (4e) uses the *merge* operations to pour tubes $T_{0,0}, T_{1,1}, T_{0,1}^{>=}, T_{0,1}^{<}, T_{1,0}^{>=}$ and $T_{1,0}^{<}$ into tube T_0. After repeating execution of Steps (4a) through (4e), it finally produces library sequences in tube T_0 that perform computation of the sum for 2 ^{n} pairs of subsets, $(A^1, \bar{A})$.

Next, Step (5) is a single loop and is mainly used to find the answer for the set-partition problem to a finite n-element set A. On each execution of Step (5a), it applies the *extract* operation to form two tubes: T_0 and T_{50}. Tube T_0 contains library sequences that have $y_{n+1,g} = 0$ and Tube T_{50} includes library sequences that have $y_{n+1,g} = 1$. Next, each execution for Step (5b) uses the *discard* operation to discard tube T_{50}. After repeating execution of Steps (5a) through (5b), it finally produces library sequences in tube T_0 that encode any answer for the set-partition problem to a finite n-element set A. Then, the execution for Step (6) employs the *detect* operation to check if tube T_0 contains any library sequence or not. If it returns a "yes," then the execution for Step (6a) uses the *read* operation to read the solution for the set-partition problem to a finite n-element set, A. Therefore, any solution for the set-partition problem to a finite n-element set A can be computed from those steps in **Algorithm 1**. ■

O. The Complexity of Solving the Set-Partition Problem to a Finite N-element Set

Theorem 2: The set-partition problem for a finite n-element set A can be solved with $O(n^*k)$ biological operations, $O(2^n)$ library sequences, $O(1)$ tubes, and the longest library strand $O(n^*k)$ from solution space of library sequences, where the number of bits for the value of each element in A is $(k+1)$ bits.

Proof: Refer to **Algorithm 1**. ■

IV. Conclusion

In this paper, the *first* DNA algorithm of a *signed* parallel adder, the *first* DNA algorithm of a *signed* parallel subtractor, and the *first* DNA algorithm of a *signed* parallel comparator are proposed to perform the function of signed parallel addition, the function of signed parallel subtraction, and the function of signed parallel comparator. Currently the future of molecular computers is unclear. It is possible that in the future molecular computers will be the clear choice for performing massively parallel computations. However, there are still many technical difficulties to overcome before this becomes a reality. We hope that this paper helps to demonstrate that molecular computing is a technology worth pursuing.

References

[1] L. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, pp. 1021–1024, 1994.
[2] W.-L. Chang, M. Guo, and M. Ho, "Fast parallel molecular algorithms for DNA-based computation: Factoring integers," *IEEE Trans. Nanobiosci.*, vol. 4, no. 2, pp. 149–163, Jun. 2005.
[3] W.-L. Chang, M. Ho, and M. Guo, "Molecular solutions for the subset-sum problem on DNA-based supercomputing," *BioSystems*, vol. 73, no. 2, pp. 117–130, 2004.
[4] W.-L. Chang and M. Guo, "Solving the set-cover problem and the problem of exact cover by 3-sets in the Adleman-Lipton model," *BioSystems*, vol. 72, no. 3, pp. 263–275, 2003.
[5] G. Paun, G. Rozenberg, and A. Salomaa, *DNA Computing: New Computing Paradigms*. New York: Springer-Verlag, 1998.
[6] L. M. Adleman, R. S. Braich, C. Johnson, P. W. K. Rothemund, D. Hwang, and N. Chelyapov, "Solution of a 20-variable 3-SAT problem on a DNA computer," *Science*, vol. 296, no. 5567, pp. 499–502, 2002.
[7] M. Guo, W.-L. Chang, M. Ho, J. Lu, and J. Cao, "Is optimal solution of every NP-complete or NP-hard problem determined from its characteristic for DNA-based computing," *Biosystems*, vol. 80, no. 1, pp. 71–82, 2005.
[8] W.-L. Chang, M. Guo, and M. Ho, "Towards solution of the set-splitting problem on gel-based DNA computing," *Future Gener. Comput. Syst.*, vol. 20, no. 5, pp. 875–885, Jun. 15, 2004.
[9] W.-L. Chang and M. Guo, "Solving the set-cover problem and the problem of exact cover by 3-sets in the Adleman-Lipton's model," *BioSystems*, vol. 72, no. 3, pp. 263–275, 2003.
[10] W.-L. Chang, M. Ho, and M. Guo, "Fast parallel molecular solution to the dominating-set problem on massively parallel bio-computing," *Parallel Comput.*, vol. 30, no. 9–10, pp. 1109–1125, 2004.
[11] W.-L. Chang, M. Guo, and J. Wu, "Solving the independent-set problem in a DNA-based supercomputer model," *Parallel Process. Lett.*, vol. 15, no. 4, pp. 469–480, Dec. 2005.
[12] C.-W. Yeh, C.-P. Chu, and K.-R. Wu, "Molecular solutions to the binary integer programming problem based on DNA computation," *Biosystems*, vol. 83, no. 1, pp. 56–66, Jan. 2006.
[13] A. Schuster, "DNA databases," *BioSystems*, vol. 81, pp. 234–246, 2005.

Weng-Long Chang received the Ph.D. degree in computer science and information engineering from National Cheng Kung University, Taiwan, R.O.C., in 1999.

He is currently an Associated Professor at National Kaohsiung University of Applied Sciences, Kaosiung, Taiwan. His research interests include the design of DNA-based algorithms on molecular computing, and languages and compilers for parallel computing.