社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－AL－97 (13)
2004／10／15

DNA計算における奇偶転換ソート及びシェアソートアルゴリズム

牛島 瑞恵† 藤原 暁宏†

† 九州工業大学 情報工学部
E-mail: †mizue@zodiac30.cse.kyutech.ac.jp, ††fujiwara@cse.kyutech.ac.jp

あらまし 近年の高性能計算に関する研究において，非シリコン型計算の1つとしてDNA分子を用いて計算を行うDNA計算が注目を集めている．本研究では，DNAを用いて表現された2進数の集合に対してソートを行うアルゴリズムを2つ提案する．これらのアルゴリズムに対する入力は，$O(mn)$ 種類の一本鎖DNAで表現されている n 個の m ビットの2進数の集合である．本研究では，まず最初に2つのソートアルゴリズムの基本演算として，2つの数を比較し昇順に並べる比較交換操作を定義し，この比較交換操作を効率よく行うアルゴリズムを提案する．このアルゴリズムの計算量は $O(n)$ 個の対の m ビットの2進数に対して $O(1)$ ステップで実行可能である．次に前述の比較交換操作を用いて奇偶転換ソートに基づくソートアルゴリズムを提案する．このアルゴリズムは，$O(mn^2)$ 種類のDNAを用いることにより $O(n)$ ステップでソートを実行可能である．最後に，前述の比較交換操作を用いてシェアソートに基づくソートアルゴリズムを提案する．このアルゴリズムは，$O(mn\sqrt{n}\log n)$ 種類のDNAを用いることにより $O(\sqrt{n}\log n)$ ステップでソートを実行可能である．

キーワード DNA計算，奇偶転換ソート，シェアソート

Odd-Even Transposition Sort and Shear Sort Algorithm with DNA strands

Mizue USHIJIMA† and Akihiro FUJIWARA†

† Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology
E-mail: †mizue@zodiac30.cse.kyutech.ac.jp, ††fujiwara@cse.kyutech.ac.jp

Abstract In recent works for high performance computing, computation with DNA molecules, that is, DNA computing, has considerable attention as one of nonsilicon based computings. In this paper, we propose two algorithm for sorting a set of binary numbers which are denoted by DNA molecules. An input of the algorithms is a set of n binary numbers of m bits which are denoted by $O(mn)$ kinds of DNA strands. In this paper, we first define a basic operation "compare and exchange", which compares and sorts two numbers in non-decreasing order. We propose an algorithm for the operation, which runs in $O(1)$ steps for $O(n)$ pairs of binary numbers. We next propose a sorting algorithm based on the odd-even transposition sort using the "compare and exchange" operation. The algorithm runs in $O(n)$ steps using $O(mn^2)$ DNA strands. We finally propose another sorting algorithm based on the shear sort using the "compare and exchange" operation. The algorithm runs in $O(\sqrt{n}\log n)$ steps using $O(mn\sqrt{n}\log n)$ DNA strands.

Key words DNA computing, Odd-even transposition sort, Shear sort

1. はじめに

近年の高性能計算に関する研究において，非シリコン型計算の1つとしてDNA分子を用いて計算を行うDNA計算が注目を集めている．DNA計算は，DNAの持つ超並列性，及び，ワトソン・クリック相補性を利用することにより，データ量が増加すると指数的に計算時間が増大するような問題に威力を発揮し，従来の計算機では指数的な計算時間を必要とする NP 完全問題を多項式的なステップ数で解くことを可能にする．

一方，DNA計算を NP 完全問題だけではなくより広い問題の解決方法として利用するためには，論理演算や算術演算のような通常の計算機が備えている基本的な演算が必要となる．このような基本演算については論理演算や算術演算についてもいくつかの研究が行われている[1]～[4], [6], [7]．

本研究ではDNA計算をより広い問題の解決方法として利用するために，DNA計算において基本操作であるソートに対する2つのアルゴリズムの提案を行う．これらのアルゴリズムに対する入力は，$O(mn)$ 種類の一本鎖DNAで表現されている n 個の m ビットの2進数の集合である．まず最初に，2つのソートアルゴリズムの基本演算として2つの数を比較し昇順に並べる比較交換操作を定義し，この比較交換操作を効率よく行うアルゴリズムを提案する．このアルゴリズムは，$O(n)$ 個の対の m ビットの2進数に対して $O(1)$ ステップで実行可能である．次に前述の比較交換操作を用いて奇偶転換ソートに基づくソートアルゴリズムを提案する．このアルゴリズムは，$O(mn^2)$ 種類のDNAを用いることにより $O(n)$ ステップでソートを実行可能である．最後に，前述の比較交換操作を用いてシェアソートに基づくソートアルゴリズムを提案する．このアルゴリズムは，$O(mn\sqrt{n}\log n)$ 種類のDNAを用いることにより $O(\sqrt{n}\log n)$ ステップでソートを実行可能である．

2. 準　　備

2.1 DNAによる符号化

本研究では，Σ を異なる記号の集合 $\{\sigma_0, \sigma_1, \ldots, \sigma_{m-1}, \overline{\sigma_0}, \overline{\sigma_1}, \ldots, \overline{\sigma_{m-1}}\}$ とし，Σ の各要素がDNAを表現するものと仮定する．ここで，σ_i と $\overline{\sigma_i}(0 \leq i \leq m-1)$ は相補的な一本鎖DNAの対を表し，これらの互いに相補的な一本鎖DNAはワトソン・クリック相補性により分子間力によって引き付け合い水素結合する．相補的な一本鎖DNAの対が，ワトソン・クリック相補性により水素結合したDNAを二本鎖DNAと呼び，$\left[\frac{\sigma_i}{\sigma_i}\right]$ と表現する．任意の核酸塩基配列からなる一本鎖DNAが生化学的に合成できることにより，2つの一本鎖DNA σ_0，σ_1 が共有結合により連結した一本鎖DNA $\sigma_0\sigma_1$ もまた合成できる．また，2つの一本鎖DNAの1部分が相補的になっているとき，2つの一本鎖DNAの1部分が結合した二本鎖DNAを形成する．例えば，$\sigma_0\sigma_1, \overline{\sigma_1\sigma_0}$ という2種類の一本鎖DNAの場合，二本鎖DNA $\left[\frac{\sigma_0\,\sigma_1}{\sigma_1\sigma_0}\right]$，$\left[\frac{\sigma_0\,\sigma_1}{\sigma_1\sigma_0}\right]$ を形成することができる．ここで使用する一本鎖DNA，及び，二本鎖DNAの集合を試験管と呼ぶ．

2.2 DNA計算モデル

本研究では，Rief [8] によって提唱されたDNA計算モデルであるRDNAモデルを使用する．RDNAモデルに基づき，DNAの基本操作として以下の8操作を用いる．

（1） $Merge(T_1, T_2)$：試験管 T_1 と試験管 T_2 のDNAを混合し，2つの集合の和を試験管 T_1 とする．

（2） $Copy(T_1, T_2)$：試験管 T_1 のDNAを複製し，同じ内容の試験管 T_2 を作成する．

（3） $Detect(T)$：試験管 T にDNAが存在するならば真を，存在しないならば偽を返す．

（4） $Separation(T_1, X, T_2)$：試験管 T_1 から集合 X に含まれる記号列を含む一本鎖DNAを試験管 T_2 に取り出す．

（5） $Selection(T_1, L, T_2)$：試験管 T_1 から長さ L のDNAを試験管 T_2 に取り出す．ここで長さ L は記号の個数とする．例えば $\sigma_0\sigma_1\sigma_2$ で表現されるDNAの長さは3とする．

表1　DNAに対する操作の例

	操作	試験管 T	試験管 T'
0	———	$\{\sigma_0\sigma_1, \overline{\sigma_0}, \overline{\sigma_1}\}$	ϕ
1	$Annealing(T)$	$\left\{\left[\frac{\sigma_0\sigma_1}{\overline{\sigma_0\sigma_1}}\right]\right\}$	ϕ
2	$Denaturation(T)$	$\{\sigma_0\sigma_1, \overline{\sigma_0\sigma_1}\}$	ϕ
3	$Copy(T, T')$	$\{\sigma_0\sigma_1, \overline{\sigma_0\sigma_1}\}$	$\{\sigma_0\sigma_1, \overline{\sigma_0\sigma_1}\}$
4	$Annealing(T)$	$\left\{\left[\frac{\sigma_0\sigma_1}{\overline{\sigma_0\sigma_1}}\right]\right\}$	$\{\sigma_0\sigma_1, \overline{\sigma_0\sigma_1}\}$
5	$Cleavage(T, \sigma_0\sigma_1)$	$\left\{\left[\frac{\sigma_0}{\overline{\sigma_0}}\right], \left[\frac{\sigma_1}{\overline{\sigma_1}}\right]\right\}$	$\{\sigma_0\sigma_1, \overline{\sigma_0\sigma_1}\}$
6	$Denaturation(T)$	$\{\sigma_0, \sigma_1, \overline{\sigma_0}, \overline{\sigma_1}\}$	$\{\sigma_0\sigma_1, \overline{\sigma_0\sigma_1}\}$
7	$Merge(T, T')$	$\{\sigma_0, \sigma_1, \sigma_0\sigma_1, \overline{\sigma_0}, \overline{\sigma_1}, \overline{\sigma_0\sigma_1}\}$	ϕ
8	$Separation(T, \{\sigma_0, \overline{\sigma_0}\}, T')$	$\{\sigma_1, \overline{\sigma_1}\}$	$\{\sigma_0, \sigma_0\sigma_1, \overline{\sigma_0}, \overline{\sigma_0\sigma_1}\}$
9	$Empty(T)$	ϕ	$\{\sigma_0, \sigma_0\sigma_1, \overline{\sigma_0}, \overline{\sigma_0\sigma_1}\}$
10	$Selection(T', 1, T)$	$\{\sigma_0, \overline{\sigma_0}\}$	$\{\sigma_0\sigma_1, \overline{\sigma_0\sigma_1}\}$

（8の後に $Detect(T)$ を行った場合は真を，9の後に $Detect(T)$ を行った場合は偽を返す．ϕ は試験管が空であることを示す．）

（6） $Cleavage(T_1, \sigma_0\sigma_1)$：試験管 T_1 において，DNAの二本鎖部分 $\left[\frac{\sigma_0\sigma_1}{\sigma_0\sigma_1}\right]$ を $\left[\frac{\sigma_0}{\sigma_0}\right]$ と $\left[\frac{\sigma_1}{\sigma_1}\right]$ に切断する．

（7） $Annealing(T)$[(注1)]：試験管 T_1 内の相補的な一本鎖DNA同士を結合させて二本鎖DNAにする．

（8） $Denaturation(T)$：試験管 T_1 内の二本鎖DNAを切り離し2つの一本鎖DNAに分解する．

上記の8操作に対して，本研究では以下の操作を加えた．

9. $Empty(T)$：試験管 T を空にする．

これらの基本操作の例を表1に示す．これらの操作は定数回の生化学的操作により実行できるとみなせる．以下では，DNAに対する1回の基本操作に要する時間を $O(1)$ ステップと呼ぶ．

2.3 DNAによる2進数の表現

本節では，DNAによる2進数の表現を定義する．文献[2]において，$O(mn)$ 種類の一本鎖DNAを用いることにより，n 個の m ビットの2進数を表現する方法が定義されている．文献[2]の方法は，n 個の2進数を1ビット毎に一本鎖DNAで表現するというアイデアに基づいており，本研究では，同様の方法でDNAにより数を表現するものとする．以下にその概要を示す．

RDNAモデルにおけるすべてのDNAは集合 Σ により表現されるので，集合 Σ の要素を組み合わせることにより数を表現する．数の表現には，以下の集合により定義される $O(m+n)$ 種類の一本鎖DNAを使用する．

（注1）：文献[8]では核酸塩基配列の間の隙間を埋める $Ligation$ という操作も定義されているが，本論文では $Annealing$ がその操作を含むものとする．

$$\Sigma = \{A_0, A_1, \ldots A_{n-1}, B_0, B_1, \ldots, B_{m-1},$$
$$C_0, C_1, D_0, D_1, 0, 1, \#, \overline{A_0}, \overline{A_1}, \ldots, \overline{A_{n-1}},$$
$$\overline{B_0}, \overline{B_1}, \ldots, \overline{B_{m-1}}, \overline{C_0}, \overline{C_1}, \overline{D_0}, \overline{D_1}, \overline{0}, \overline{1}, \overline{\#}\}$$

ここで，$A_0, A_1, \ldots, A_{n-1}$ はアドレスを表し，$B_0, B_1, \ldots, B_{m-1}$ はビットを表すものとする．C_0, C_1 及び D_0, D_1 は *Cleavage* により切断する部分を表し，二本鎖 DNA $\left[\frac{C_0C_1}{C_0C_1}\right]$ は $\left[\frac{C_0}{C_0}\right]\left[\frac{C_1}{C_1}\right]$ のように，二本鎖 DNA $\left[\frac{D_0D_1}{D_0D_1}\right]$ は $\left[\frac{D_0}{D_0}\right]\left[\frac{D_1}{D_1}\right]$ のように切断されるものとする．"0" 及び"1" はビットの値を表し，"#" は *Separation* で用いるための記号とする．

本研究では，アドレス i の j 番目のビットの値が $V_{i,j}(\in \{0,1\})$ の場合は，

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

と表現するものとする．ここで，$S_{i,j}$ をメモリ鎖と呼び，n 個の m ビットの 2 進数は $O(mn)$ 種類の一本鎖 DNA で表現する．つまり，アドレス i の値が $V_i = \sum_{j=0}^{m-1} V_{i,j} \times 2^j$ であるような 2 進数 $V_{i,m-1}V_{i,m-2}\ldots V_{i,0}$ は，$\{S_{i,m-1}, S_{i,m-2}, \ldots, S_{i,0}\}$ の $O(m)$ 種類のメモリ鎖の集合で表現する．また，本研究では各 2 進数の最上位ビットである $m-1$ 番目のビットは符号を表すビットとし，負の値は 2 の補数表現を用いて表されるものとする．

なお，本研究では値を保持するメモリ鎖を

$$S_{i,j}(0) = D_1 A_i B_j C_0 C_1 0 D_0$$
$$S_{i,j}(1) = D_1 A_i B_j C_0 C_1 1 D_0$$

のように簡略化して表現する．

2.4 奇偶転換ソート

本節では，既知の並列アルゴリズムである奇偶転換ソートアルゴリズム [5] について説明する．奇偶転換ソートを用いると，プロセッサ数 n のライン上で n 個の数を n ステップでソートできるということが示されている．ここでラインとは，隣接するプロセッサのみと接続されている分散メモリ型並列計算モデルのことである．各プロセッサには，0 から $n-1$ までのアドレスが，左端から右端へ向かって割り当てられている．

このライン上での奇偶転換ソートの概要を以下に示す．奇偶転換ソートは，2 つの値 x, y を比較し，$x = min\{x, y\}$, $y = max\{x, y\}$ という代入を行う操作を基本操作とする．以下ではこの操作を比較交換操作と呼ぶ．このアルゴリズムの特徴は，奇数ステップと偶数ステップでは比較交換操作の対象となる対が異なるという点にある．n 個の数 (それぞれの数は $0, 1, \ldots, n-1$ のアドレスに保持されている) のソートを行うために，偶数ステップでは $2i$ と $2i+1$ ($0 \leqq i \leqq \frac{n}{2}-1$) の，奇数ステップでは $2i-1$ と $2i$ ($1 \leqq i \leqq \frac{n}{2}-1$) のアドレス対の数に対して比較交換操作を行う．この操作を 0 ステップから $n-1$ ステップまで繰り返すことにより，全体のソートを行うことができる．図 1 に入力サイズが 6 の場合の奇偶転換ソートの例を示す．

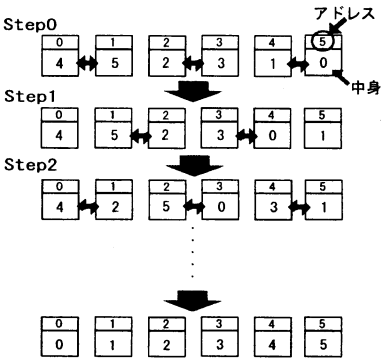

図 1　奇偶転換ソートの例 ($n = 6$ の場合)

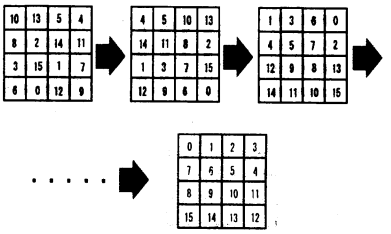

図 2　シェアソートの例 ($n = 16$ の場合)

2.5 シェアソート

本節では，既知の並列アルゴリズムであるシェアソートアルゴリズム [9], [10] について説明する．シェアソートを用いると，$\sqrt{n} \times \sqrt{n}$ のメッシュ上で n ($n = x^2 \mid x$ は整数) 個の数を $\log n + 1$ ステップでソートできるということが示されている．ここでメッシュとは，2 次元格子状に配置した各ノードが 4 個の隣接ノードと接続されている結合網のことである．

このメッシュ上でのシェアソートの概要を以下に示す．シェアソートは偶数ステップと奇数ステップでソートする部分が異なり，偶数ステップでは行に対して，奇数ステップでは列に対して奇偶転換ソートを実行する．列の場合は，どの列も上の方が小さくなるようにソートし，行の場合は，図 2 のように，偶数行では左の方が小さく，奇数行では右の方が小さくなるようにソートする．この操作を 0 ステップから $\log n$ ステップまで繰り返すことにより，全体のソートを行うことができる．図 2 にサイズが $n = 16$ の場合の例を示す．

3. 比較交換操作を行うアルゴリズム

奇偶転換ソートでは，2 つの値を比較し交換するという比較交換操作を並列に繰り返す．そこで奇偶転換ソートについて考える前に，この比較交換操作を行うアルゴリズムの提案を行う．

3.1 アルゴリズムの概要

まず最初に，比較交換操作を行う 2 つの数を $A_i(x), A_j(y)$ と仮定する．ここで，x, y は 2 つの値を表し，i, j は x, y が格納されているアドレスをそれぞれ表している．したがって，$A_i(x)$ はアドレス i に値 x を格納しているメモリ鎖の集合を表し，

$A_j(y)$ はアドレス j に値 y を格納しているメモリ鎖の集合を表している．

アルゴリズムの最初のステップでは，入力された値のアドレスの中身を交換した $A_i(y), A_j(x)$ を作成する．以下ではこの操作を交換操作と呼ぶ．アルゴリズムでは，減算により2つの値の大小を比較する．この比較結果より2つの値がアドレスに対して昇順に並んでいなければ，交換操作を実行して得られるメモリ鎖を出力し，そうでなければ入力された値をそのまま出力する．

例として，8，3という2つの値が入力された場合を考える．8はアドレス0に，3はアドレス1に格納されており，それぞれを $A_0(8)$，$A_1(3)$ のように表す．$A_0(8)$ と $A_1(3)$ を比較するために $3-8$ の減算を行う．減算結果の-5はアドレス0に格納され，その符号ビット（$m-1$ 番目のビット）は負である．このことから入力された2つの値は昇順に並んでいないことが分かる．そこで，$A_0(8)$ と $A_1(3)$ に対して交換操作を実行して得られる $A_0(3)$，$A_1(8)$ というメモリ鎖を用意する．比較した結果より，先ほど用意した $A_0(3)$，$A_1(8)$ をソートの結果として出力する．

以下に DNA に対する操作を用いたアルゴリズムの概要を示す．このアルゴリズムの入力は $\frac{n}{2}$ 個の2進数の対 (V_{2i}, V_{2i+1}) $(0 \leqq i \leqq \frac{n}{2}-1)$ であり，各2進数は m ビットで構成される．まず最初に，アルゴリズムで使用される試験管の役割を示す．

(試験管)

- T_{input}：入力された数を表す DNA 鎖を保持
- T_{output}：出力される数を表す DNA 鎖を保持
- T_{sub}：減算結果を表す DNA 鎖を保持
- T_{sign}：数の符号を表す DNA 鎖を保持
- T_{lose}：減算結果が負の数になった対を保持
- T_{change}：入力値に対し交換操作を実行して得られる DNA 鎖を保持
- T_{tmp}：一時的に使用
- T_{trash}：不要な DNA 鎖を入れる

以下にアルゴリズムの概要を示す．このアルゴリズムは大きく分けて4ステップで構成されている．

Step 1： T_{input} を T_{output} にコピーする．

Step 2： 数の比較（減算）を行う．
$R = \{(A_{2i+1}, A_{2i})\}$ に対して減算 $V_{2i+1} - V_{2i}$ を実行する．減算結果はアドレス A_{2i} に格納し，試験管 T_{sub} に保存する．

Step 3： 減算結果の符号ビットを T_{sub} から T_{sign} に抜き出す．このとき，減算結果が負（符号ビットが1）ならば，2つの数のアドレスに印をつけて T_{lose} に保存する．

Step 4： 以下の部分操作を実行する．

(4-1) 入力値に対して交換操作を実行し，得られたメモリ鎖を T_{change} に保存する．

(4-2) T_{output} から印のついたアドレスを表す DNA 鎖を取り除く．

(4-3) T_{change} から印のついたアドレスを表す DNA 鎖を取り出し，それを T_{output} に加える．

3.2 アルゴリズムの詳細

以下では各ステップの基本操作を用いた実現方法について述べる．

Step 1は，基本操作 *Copy* を用いることにより $O(1)$ ステップで実行可能である．

Step 2では，減算を行うアドレス対を保持する試験管 $R = \{(A_{2i+1}, A_{2i})\}$ に対して減算 $V_{2i+1} - V_{2i}$ を実行する．その結果はアドレス A_{2i} に代入されて T_{sub} に格納されるが，T_{sub} の内容は，

$$T_{sub} = \{S_{2i,j} \mid V_{2i} = V_{2i+1} - V_{2i},\ 0 \leqq i \leqq \frac{n}{2}-1,\ 0 \leqq j \leqq m-1\}$$

となる．この減算は *SubtractionOperation* [2] によって $O(1)$ ステップで実行可能である．

Step 3では，まず最初に，基本操作 *Separation* により減算結果の符号ビット $S_{2i,m-1}$ を T_{sub} から T_{sign} に $O(1)$ ステップで抜き出す．次に，以下のように定義される T_{lose} を用意する．

$$T_{lose} = \{D_1, \overline{S_{i,m-1}(1)D_1\#D_0D_1A_i},\ \overline{S_{i,m-1}(1)D_1\#D_0D_1A_{i+1}} \mid 0 \leqq i \leqq n-2\}$$

このような T_{lose} に対して T_{sign} を基本操作 *Merge* によりマージし，基本操作 *Annealing*，*Cleavage*，*Denaturation* を実行することにより，減算結果が負となる対のアドレスに，印として記号"#"を付けることができる．

これら以外の DNA 鎖は不要なので，これらを取り除くために，一本鎖 DNA の集合 $\{C_0C_1, \overline{C_0C_1}, D_1\}$ を用いて基本操作 *Separation* を実行すると，T_{lose} の内容は，

$$T_{lose} = \{\overline{D_1\#D_0D_1A_{2i}},\ \overline{D_1\#D_0D_1A_{2i+1}} \mid V_{2i+1} < V_{2i},\ 0 \leqq i \leqq \frac{n}{2}-1\}$$

となる．

(4-1)は，基本操作 *Copy* と *LogicOperation* [2] を用いることで $O(1)$ ステップで実行可能である．この場合，*LogicOperation* で実行される論理演算は，表2の真理値表で定義され，試験管 L の中身は

$$L = \{\overline{0\#D_0S_{2i,j}(0)S_{2i+1,j}(0)D_10\#},\ \overline{1\#D_0S_{2i,j}(0)S_{2i+1,j}(1)D_10\#},\ \overline{0\#D_0S_{2i,j}(1)S_{2i+1,j}(0)D_11\#},\ \overline{1\#D_0S_{2i,j}(1)S_{2i+1,j}(1)D_11\#} \mid 0 \leqq i \leqq \frac{n}{2}-1\}$$

となる．また，その結果を T_{change} に保存する．

(4-2)では，まず T_{lose} に一本鎖 DNA $\{D_1\#D_0\}$ を基本操作 *Merge* を用いてマージする．ここで，T_{lose} は(4-3)でも必要なので，基本操作 *Copy* により T'_{lose} に複製を作成しておく．その後，T_{output} に T_{lose} を基本操作 *Merge* によりマージし，

表 2　L の表す論理演算

入力		出力	
V_{2i}	V_{2i+1}	V_{2i}	V_{2i+1}
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

基本操作 *Annealing*, *Denaturation* を実行すると，取り除きたい DNA 鎖の先頭に $D_1\#D_0$ が付くので，これらを取り除くために $\{\#, \overline{\#}\}$ を用いて基本演算 *Separation* を実行する．これにより，(4-2) は $O(1)$ で実行可能である．

(4-3) では，最初に T_{change} に T'_{lose} を基本操作 *Merge* を用いてマージし，基本操作 *Annealing*, *Denaturation* を実行すると，取り出したい DNA 鎖の先頭に $D_1\#D_0$ が付く．次に $\{\#\}$ を用いて基本演算 *Separation* を実行し，取り出したい DNA 鎖のみを T_{tmp} に取り出す．これより T_{tmp} の内容は，

$$T_{tmp} = \{D_1\#D_0S_{2i,j}\ ,\ D_1\#D_0S_{2i+1,j} \mid V_{2i+1} < V_{2i}\ ,\ 0 \leqq i \leqq \frac{n}{2} - 1\ ,\ 0 \leqq j \leqq m-1\}$$

となる．T_{tmp} に $\overline{D_0D_1}$ を基本操作 *Merge* によりマージし，基本操作 *Annealing*, *Cleavage*, *Denaturation* を実行すると，$D_1\#D_0$ とメモリ鎖に切り離される．$D_1\#D_0$ など余分な DNA を取り除くために $\{\#, \overline{D_0}, \overline{D_1}\}$ を用いて基本操作 *Separation* を実行する．最後に，T_{tmp} を T_{output} に基本操作 *Merge* を用いてマージすることで，入力値に対して比較交換操作を実行した結果を T_{output} に保存しソートが完了する．

以下に基本操作を用いたアルゴリズムの詳細をまとめる．

Procedure
Compare_and_Exchange$(T_{input}, R, L, T_{output})$\{
/* Step 1 */
Copy(T_{input}, T_{output});
/* Step 2：2 つの値の比較のため
減算 $(V_{2i} = V_{2i+1} - V_{2i})$ を行う */
SubtractionOperation(T_{output}, R, T_{sub});
/* Step 3：減算結果の最上位ビットを取り出す */
Separation$(T_{sub}, \{B_{m-1}\}, T_{sign})$;
/* 減算結果が負ならば印として"#" をつける */
Merge(T_{lose}, T_{sign});
Annealing(T_{lose});
Cleavage(T_{lose}, D_0D_1);
Denaturation(T_{lose});
Separation$(T_{lose}, \{C_0C_1, \overline{C_0C_1}, D_1\}, T_{trash})$;
/* Step 4 */
/* (4-1)：入力値に対して交換操作を実行 */
Copy(T_{output}, T_{change});
LogicOperation$(T_{change}, L, T_{change})$;
/* (4-2)：T_{output} から
"#" のついた DNA 鎖を取り除く */
Merge$(T_{lose}, D_1\#D_0)$;
Copy(T_{lose}, T'_{lose});
Merge(T_{output}, T_{lose});
Annealing(T_{output});
Denaturation(T_{output});
Separation$(T_{output}, \{\#, \overline{\#}\}, T_{trash})$;
/* (4-3)：T_{change} から
"#" のついた DNA 鎖を取り出す */
Merge(T_{change}, T'_{lose});
Annealing(T_{change});
Denaturation(T_{change});
Separation$(T_{change}, \#, T_{tmp})$;
/* 取り出した DNA 鎖を T_{output} に加える */
Merge$(T_{tmp}, \overline{D_0D_1})$;
Annealing(T_{tmp});
Cleavage(T_{tmp}, D_0D_1);
Denaturation(T_{tmp});
Separation$(T_{tmp}, \{\#, \overline{D_0}, \overline{D_1}\}, T_{trash})$;
Merge(T_{output}, T_{tmp});
\}

このアルゴリズムの各ステップは $O(1)$ ステップで実行可能である．また，このアルゴリズムで必要な DNA の種類は $O(mn)$ 種類である．これらのことから次の定理を得ることができる．

[定理 1]　$O(n)$ 個の対の m ビットの 2 進数に対して比較交換操作を行うアルゴリズム *Compare_and_Exchange* は，$O(mn)$ 種類の DNA を用いることにより $O(1)$ ステップで実行可能である．□

4. 奇偶転換ソートアルゴリズム

ここでは DNA を用いた奇偶転換ソートアルゴリズムの実現について，説明を行う．

4.1 アルゴリズムの概要

n 個の数のソートを $O(n)$ ステップで実行するアルゴリズムの概要を以下に示す．なお，簡単のため，n は偶数であると仮定する．このアルゴリズムでは，2 つの値の比較交換操作に前述したアルゴリズム *Compare_and_Exchange* を使用する．まず入力された数に対して，偶数ステップ，奇数ステップで異なるアドレス同士の比較交換操作を実行する．この操作を n 回繰り返すことによりソートが完了する．

例として，34，29，6，10，16，8 という 6 個の数が入力されたとする．それぞれの数はアドレス $A_0, A_1, \ldots, A_5$ に格納されており，それぞれを $A_0(34), A_1(29), \ldots, A_5(8)$ のように表す．0 ステップ目では，$A_0(34)$ と $A_1(29)$，$A_2(6)$ と $A_3(10)$，$A_4(16)$ と $A_5(8)$ を比較する．それぞれの対に対して交換操作を実行し，$A_0(29)$ と $A_1(34)$，$A_2(10)$ と $A_3(6)$，$A_4(8)$ と $A_5(16)$ のようなメモリ鎖を作成する．$A_0(34)$ と $A_1(29)$，$A_4(16)$ と $A_5(8)$ は交換が必要なので取り除き，それぞれ $A_0(29)$ と $A_1(34)$，$A_4(8)$ と $A_5(16)$ と交換する．すると入力された数は $A_0(29), A_1(34), A_2(6), A_3(10), A_4(8), A_5(16)$ となる．図 3 にこの交換の様子を示す．1 ステップ目では，$A_1(34)$ と $A_2(6)$，

$A_3(10)$ と $A_4(8)$ を比較する．どちらも交換が必要なので，交換操作により得られたメモリ鎖と交換する．このようにして比較交換操作を n 回繰り返すとソートが完了する．

以下に DNA に対する操作を用いたアルゴリズムの概要を示す．まず最初に，アルゴリズムで使用される試験管の役割を示す．

(試験管)

- T_{input}：入力された数を表す DNA 鎖を保持
- T_{output}：出力される数を表す DNA 鎖を保持

以下にアルゴリズムの概要を示す．

Step 1： T_{input} を T_{output} にコピーする．

Step 2： 各 j 番目のステップ $(0 \leqq j \leqq n-1)$ において，

j が偶数ならば，

$R_{even} = \{(A_{2i+1}, A_{2i}) \mid 0 \leqq i \leqq \frac{n}{2} - 1\}$，

j が奇数ならば，

$R_{odd} = \{(A_{2i}, A_{2i-1}) \mid 1 \leqq i \leqq \frac{n}{2} - 1\}$，

に対して，比較交換操作を並列に実行する．

4.2 アルゴリズムの詳細

以下では各ステップの基本操作を用いた実現方法について述べる．Step 1 は，基本操作 *Copy* を用いることにより $O(1)$ ステップで実行可能である．Step 2 では，比較交換操作を並列に実行する．これは，前述した処理 *Compare_and_Exchange* を用いることで $O(1)$ ステップで実行可能であり，このステップを n 回繰り返す．また *Compare_and_Exchange* 内の論理演算時には，偶数ステップでは L_{even}，奇数ステップでは L_{odd} に従って論理演算を実行する．L_{even}, L_{odd} はそれぞれ表 3，表 4 の真理値表によりそれぞれ定義され，その中身は

$$L_{even} = \{\overline{0\#D_0S_{2i+1,j}(0)S_{2i,j}(0)D_10\#},\ \overline{1\#D_0S_{2i+1,j}(0)S_{2i,j}(1)D_10\#},\ \overline{0\#D_0S_{2i+1,j}(1)S_{2i,j}(0)D_11\#},\ \overline{1\#D_0S_{2i+1,j}(1)S_{2i,j}(1)D_11\#} \mid 0 \leqq i \leqq \frac{n}{2} - 1\}$$

$$L_{odd} = \{\overline{0\#D_0S_{2i,j}(0)S_{2i-1,j}(0)D_10\#},\ \overline{1\#D_0S_{2i,j}(0)S_{2i-1,j}(1)D_10\#},\ \overline{0\#D_0S_{2i,j}(1)S_{2i-1,j}(0)D_11\#},\ \overline{1\#D_0S_{2i,j}(1)S_{2i-1,j}(1)D_11\#} \mid 0 \leqq i \leqq \frac{n}{2} - 1\}$$

の様になっている．

以下に基本操作を用いたアルゴリズムの詳細をまとめる．

```
Procedure
OddEvenTranspositionSort
(T_input, R_even, R_odd, L_even, L_odd, T_output) {
  /* Step 1 */
  Copy(T_input, T_output);
  for (ST = 0; ST ≦ n − 1; ST++) {
    /* Step 2：比較交換操作を並列に実行する．
       比較する対は ST の偶奇により異なる */
    if (ST%2 == 0) {
      Compare_and_Exchange
      (T_input, R_even, L_even, T_output);
    }
    else{
      Compare_and_Exchange
      (T_input, R_odd, L_odd, T_output);
    }
  }
}
```

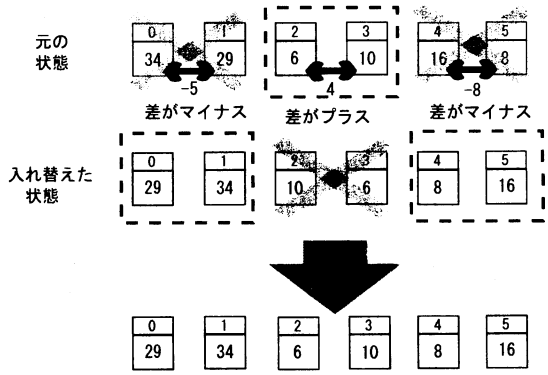

図 3 奇偶転換ソートにおける比較，交換の例

表 3 L_{even} の表す論理演算

入力		出力	
V_{2i}	V_{2i+1}	V_{2i}	V_{2i+1}
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

表 4 L_{odd} の表す論理演算

入力		出力	
V_{2i-1}	V_{2i}	V_{2i+1}	V_{2i}
0	0	0	0
0	1	1	0
1	0	0	1
1	1	1	1

このアルゴリズムでは Step 2 を n 回繰り返しているが，Step 2 は $O(1)$ ステップで実現可能なので，このアルゴリズムは $O(n)$ ステップで実行可能である．また，このアルゴリズムで必要な DNA の種類は $O(mn^2)$ 種類である．これらのことから次の定理を得ることができる．

[定理 2] n 個の m ビットの 2 進数のソートを行うアルゴリズム *OddEvenTranspositionSort* は，$O(mn^2)$ 種類の DNA を用いることにより $O(n)$ ステップで実行可能である． □

5. シェアソートアルゴリズム

ここでは DNA を用いたシェアソートアルゴリズムの説明を

行う．このアルゴリズムにおいて，それぞれの行，列のソートには奇偶転換ソートを用いる．なお，簡単のため，ある整数 x に対して $n = x^2$ であるとする．行のソートを簡単にするため，n 個の数それぞれのアドレス $A_0, A_1, \ldots, A_{n-1}$ を図 4 の様に $\sqrt{n} \times \sqrt{n}$ のメッシュ上に蛇状配置する．それぞれの行，列をソートするためにはソートを実行する行または列のアドレスを入力し，そのアドレスの集合に対して奇偶転換ソートを適用すればよい．

5.1 行および列の入力アドレス

この節では，奇偶転換ソートの入力となる行，及び，列のアドレスについて考える．

まず，行のアドレスについて考える．この場合，行は 0 行目から $\sqrt{n}-1$ 行目まで合計 $\sqrt{n}$ 行存在する．偶数番目の行では左側が小さくなるようにソートするが，その行では左側から昇順にアドレスが並んでいる．反対に奇数番目の行では右側が小さくなるようにソートするが，アドレスは右側から昇順に並んでいる．これらのことから各行のアドレスを $E_k(0 \leq k \leq \sqrt{n}-1)$ とすると，

$$E_k = \{A_{k\sqrt{n}+j} \mid 0 \leq j \leq \sqrt{n}-1\}$$

と定義できる．

次に列のアドレスについて考える．行と同様，列も 0 列目から $\sqrt{n}-1$ 列目まで合計 $\sqrt{n}$ 列存在する．例として，図 4 のように，$n = 16$ の場合を考える．それぞれのアドレスの，1 つ下のアドレスとの差を考えると，最も左の列ではアドレス同士の差が $7, 1, 7, 1, \ldots$ と変化しているのが分かる．次の列では $5, 3, 5, 3, \ldots$，最も右の列では $1, 7, 1, 7, \ldots$ というように，足し合わせると $8(= 2\sqrt{n})$ となる 2 つの奇数が交互に現れている．このことから，それぞれの列のアドレス $F_k(0 \leq k \leq \sqrt{n}-1)$ は 0 行目のアドレスに 2 つの奇数を交互に足していけば求まる．よって，列のアドレスを $F_k(0 \leq k \leq \sqrt{n}-1)$ とすると，

$$F_k = \{A_{k+\lceil j/2 \rceil(\sqrt{n}-(2k+1))+\lfloor j/2 \rfloor(2k+1)} \mid 0 \leq j \leq \sqrt{n}-1\}$$

と定義できる．

2 つ目の項の $\sqrt{n}-(2k+1)$ は 0 行目と 1 行目の数の差を表し，3 つ目の項の $2k+1$ は 1 行目と 2 行目の数の差を表す．これら 2 つの数を 0 行目のアドレスを表す最初の項 k に足すことにより比較交換操作を実行するアドレスを算出できる．

5.2 アルゴリズムの概要

n 個の数のソートを $\sqrt{n} \times \sqrt{n}$ のメッシュ上で $O(\sqrt{n}\log n)$ ステップで実行するアルゴリズムの概要を以下に示す．このアルゴリズムは偶数ステップと奇数ステップでそれぞれ必要な DNA 鎖を奇偶転換ソートによってソートすることを繰り返す．

このアルゴリズムに対する入力は n 個の m ビットの 2 進数である．まず最初に，アルゴリズムで使用される試験管の役割を示す．

(試験管)

- T_{input} : 入力された数を表す DNA 鎖を保持
- T_{output} : 出力される数を表す DNA 鎖を保持

以下にアルゴリズムの概要を示す．

A_0	A_1	A_2	A_3
A_7	A_6	A_5	A_4
A_8	A_9	A_{10}	A_{11}
A_{15}	A_{14}	A_{13}	A_{12}

図 4 $n = 16$ の場合

Step 1 : T_{input} を T_{output} にコピーする．

Step 2 : 各行，及び，列をソートするため，以下の操作を $\log n + 1$ 繰り返す．

偶数ステップならばそれぞれの行のアドレスの集合 $E_k(0 \leq k \leq \sqrt{n}-1)$ に，奇数ステップならばそれぞれの列のアドレスの集合 $F_k(0 \leq k \leq \sqrt{n}-1)$ に奇偶転換ソートを適用する．

5.3 アルゴリズムの詳細

以下では各ステップの基本操作を用いた実現方法について述べる．

Step 1 は，基本操作 *Copy* を用いることにより $O(1)$ で実行可能である．Step 2 の奇偶転換ソートは，前述した処理 *OddEvenTranspositionSort* を用いることで $O(n)$ で実行可能である．

以下に基本操作を用いたアルゴリズムの詳細をまとめる．また，ここで用いている $R_{even}, R_{odd}, L_{even}, L_{odd}$ は，集合 E_k, F_k $(0 \leq k \leq \sqrt{n}-1)$ により導出され，定義されているものとする．

```
Procedure
ShearSort(T_input, R_even, R_odd, L_even, L_odd, T_output){
  /* Step 1 */
  Copy(T_input, T_output);
  for (ST = 0; ST ≦ log n; ST++) {
    /* Step 2 : それぞれの行，及び，列をソート */
    if (ST%2 == 0) {
      OddEvenTranspositionSort
      (T_output, R_even, L_even, T_output);
    }
    else{
      OddEvenTranspositionSort
      (T_output, R_odd, L_odd, T_output);
    }
  }
}
```

このアルゴリズムは Step 2 を $\log n + 1$ 回繰り返す．Step 2 において，*OddEvenTranspositionSort* でソートする要素の数は $\sqrt{n}$ なので，$O(\sqrt{n})$ ステップで実行可能であり，必要な DNA の種類は $O(mn\sqrt{n}\log n)$ 種類である．これより，次の定理を得ることができる．

[定理 3] n 個の m ビットの 2 進数のソートを行うアルゴリズム *ShearSort* は，$O(mn\sqrt{n}\log n)$ 種類の DNA を用いることにより $O(\sqrt{n}\log n)$ ステップで実行可能である． □

6. ま と め

本研究では DNA 計算において奇偶転換ソート，及び，シェアソートを実現するアルゴリズムを提案した．まず最初に 2 つのソートアルゴリズムの基本演算として 2 つの数を比較し昇順に並べる比較交換操作を効率よく行うアルゴリズムを提案した．このアルゴリズムは，$O(n)$ 個の m ビットの 2 進数に対して $O(1)$ ステップで実行可能である．この比較交換操作を元に，$O(n)$ ステップでソートを行う奇偶転換ソートアルゴリズム，及び，$O(\sqrt{n}\log n)$ ステップでソートを行うシェアソートアルゴリズムを提案した．

本研究で提案したアルゴリズムは理論に基づくものであり，実際の DNA を使用した実験は行っていない．これらのアルゴリズムを DNA コンピュータによって実現するために，生成する DNA の長さをメモリ容量やビット幅に比例しないようにすること，基本操作におけるエラー率を考慮し，計算精度の高いアルゴリズムの提案を行うことなどが今後の研究課題である．

文 献

[1] P.Frisco. Parallel arithmetic with splicing. *Romanian Journal of Information Science and Technology(ROMJIST)*, Vol. 2, No. 3, pp. 113–128, 2000.

[2] A.Fujiwara, K.Matsumoto, and W.Chen. Procedures for Logic and Arithmetic operations with DNA Molecules. *International Journal of Foundations of Computer Science*, Vol. 15, No. 3, pp. 461–474, 2004.

[3] F.Guarnieri, M.Fliss, and C.Bancroft. Making DNA add. *Science*, Vol. 273(5272), pp. 220–223, 1996.

[4] V.Gupta, S.Parthasarathy, and M.J.Zaki. Arithmetic and logic operations with DNA. *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers*, pp. 212–220, 1997.

[5] N.Haberman. Parallel neighbor-sort(or the glory of the induction principle). Technical Report AD-759 248, National Technical Information Service, US Department of Commerce, 5285 Por Royal Road, Springfield VA 22151, 1972.

[6] H.Hug and R.Schuler. DNA-based parallel computation of simple arithmetic. *Proceedings of the 7th International Meeting on DNA Based Computers(DNA7)*, pp. 159–166, 2001.

[7] Z.F.Qiu and M.Lu. Arithmetic and logic operations for DNA computers. *Proceedings of the Second IASTED International conference on Parallel and Distributed Computing and Networks*, pp. 481–486, 1998.

[8] J.H.Reif. Parallel biomolecular computation: Models and simulations. *Algorithmica*, Vol. 25, No. 2-3, pp. 142–175, 1995.

[9] K.Sado and Y.Igarashi. Some parallel sorts on a mesh-connected processor array and their time efficiency. *Journal of Parallel and Distributed Computing*, 3:398-410, 1986.

[10] I.Scherson, S.Sen, and A.Shamir. Shear-sort: A true two-dimensional sorting technique for VLSI networks. In *IEEE-ACM International Conference on Parallel Proceedings*, pages 903-908. IEEE, August 1986.