アルゴリズム 51－6
（1996．5．29）

障害のある並列コンピュータ上でのソーティングアルゴリズム

今村 洋[1]　山本 博章[1]　小林 洋[2]

[1]信州大学工学部情報工学科
[2]東海大学工学部経営工学科

あらまし： 並列コンピュータ上での実用的なソーティングアルゴリズムとして、バイトニックソートが知られている。本研究では、障害のある並列コンピュータ上で、ソーティングを効率的に行なうためのアルゴリズムを提案する。基本的に、このアルゴリズムはバイトニックソートに基づいている。従来提案されたバイトニックソートに基づいた耐故障性アルゴリズムは、$O((\log n)^2)$ の冗長ステップが必要であったが、本論文のアルゴリズムはこれを $O(\log n)$ に改善した。

Sorting algorithm on parallel computers with a node failure

Hiroshi IMAMURA[1] Hiroaki YAMAMOTO[1] Hiromi KOBAYASHI[2]

[1]Faculty of Engineering, Shinshu University
[2]Faculty of Engineering, Tokai University

Abstract: This paper is concerned with parallel sorting algorithms on parallel computer with a node failure. Bitonic sort is well known as an efficient parallel sorting algorithm. However, it does not work if a parallel computer has a failure node. We here present a new algorithm based on bitonic sort for such a parallel computer with a node failure. This algorithm is more efficient than existing algorithms.

1　まえがき

多数のプロッセサをネットワーク結合した超並列コンピュータ上での並列ソーティングアルゴリズムを考える。このような、システムにおいては1つのプロッセサの故障が、処理全体に多大な影響を与えることがある。

ソーティングに関する耐故障性アルゴリズムに関しては従来よりいくつか研究されている。例えば、Sun [7] は比較器からなるソーティングネットワークを考え、その上で、耐故障性を持つソーティングネットワークの設計に関する研究を行なっている。ここでは、いくつかの種類の比較器の故障を定義し、その上で正常にソーティングを行なうソーティングネットワークを与えている。

本研究では、ネットワーク結合した並列コンピュータを考え、その上での耐故障性並列ソーティングアルゴリズムを考える。我々は、この線に沿って、今までに、バイトニックソートを基本とした研究を行なってきた [3, 4, 5, 6]。ここでは従来の改良型を示す。

バイトニックソートは代表的な並列ソーティングアルゴリズムであり、n 個のデータを $O((\log n)^2)$ 時間でソートできる。例えば、0から順番に番号づ

けられた n 個のプロセッサからなるネットワーク型並列コンピュータを考える。各プロセッサはデータを持つ。そのとき、バイトニックソートのアルゴリズムに従い、互いにデータを比較、交換し合うことにより、最終的に、プロセッサの番号順にデータを効率良くソートできる。しかしながら、プロセッサが故障している場合、バイトニックソートのアルゴリズムをそのまま適用したのでは、ソートすることができない。ここで、プロセッサが故障しているとは、完全に他のプロセッサと通信不能であることを意味する。

このような場合に対処するために、バイトニックソートに基づいたアルゴリズムが提案されている。そこでは、故障プロセッサを除いた残りのプロセッサだけを利用し、バイトニックソートにわずかな冗長ステップを付け加えることにより、正常に動作するプロセッサ間でのソーティングを可能にしている。例えば、小林 [5, 6] は 1 ノードが故障している場合、$O((\log n)^2)$ の冗長ステップを加えることにより、ソーティング可能であることを示した。また、[4] では番号 0 を持つプロセッサが故障している場合はバイトニックソートがそのまま適用できることを示している。

本論文では小林 [5, 6] の結果を改良したアルゴリズムを示す。すなわち、[4] の結果を利用することにより、従来のアルゴリズムは、バイトニックソートの時間に加え、$O((\log n)^2)$ の冗長ステップ必要であった。しかし、本論文で提案するアルゴリズムは、この冗長ステップが高々$O(\log n)$ である。さらに、この冗長ステップは、バイトニックソートと同じ構造をしているため、特定のネットワーク構造 (例えば、メッシュ、ハイパーキューブなど) をもつ並列コンピュータへ適用したとき、バイトニックソートと同じ解析ができる。

本論文の構成は以下の様になっている。2 節で、基本的な定義およびバイトニックソートの簡単な説明を与える。3 節で、アルゴリズムを与え、4 節でその証明を与える。

2 バイトニックソート

2.1 バイトニックソートの原理

バイトニックソートは双調列 (bitonic sequence) の性質を利用したソート法である。列 $a_1, a_2, \cdots, a_{2n}$ は次の二つの条件のいずれかを満たすとき、双調列である。

1. $a_1 \leq \cdots \leq a_{i-1} \leq a_i \geq a_{i+1} \geq \cdots \geq a_{2n}$ となるような整数 $1 \leq i \leq 2n$ が存在する。
2. 最初は 1 の条件を満たさなくても、巡回シフトを繰り返すことによって 1 の条件が満たされる。

この時、双調列に対し次の性質が成り立つ。

[性質]

$a_1, a_2, \cdots, a_{2n}$ の双調列から次の二つの列を作る。

$$A = \{\min(a_1, a_{n+1}), \cdots, \min(a_n, a_{2n})\}$$
$$B = \{\max(a_1, a_{n+1}), \cdots, \max(a_n, a_{2n})\}$$

すると次のことが成り立つ。

1. A、B は共に双調列である。
2. A の列の要素はどれも B の列のどの要素よりも小さい。

上の性質は、$\{a_1, a_2, \cdots, a_{2n}\}$ の双調列を次のようにして昇順にソートできることを意味する。

$\{a_1, a_2, \cdots, a_{2n}\}$ の双調列から二つの部分列

$$\min(a_1, a_{n+1}), \min(a_2, a_{n+2}), \cdots, \min(a_n, a_{2n})$$

および

$$\max(a_1, a_{n+1}), \max(a_2, a_{n+2}), \cdots, \max(a_n, a_{2n})$$

を作る。

min 側の各要素は max 側の各要素を越えることはなく、これら二つの部分列はそれぞれ双調列であるから、新しくできた二つの列からさらに min 側、max 側の列を作る。

以上のことを再帰的に用いて、データ数 $n = 2^r$ のとき、双調列の長さを $2^2, 2^3, \cdots, 2^r$ としていき、

最終的に 2^r の単調列を作ることでソーティングを行なう。

n 個のデータを n 個のプロセッサでバイトニックソートする場合のステップ数は $O((\log n)^2)$ となる。ここでのソーティングとは以下の処理を意味する。各ノードプロセッサは 0 から $n = 2^r - 1$ の数で番号づけされおり、データを一つ持つ。ソーティングとは各ノードが持つデータをノードの番号順に並べ換える処理である。

$n = 8$ のときのバイトニックソートによる処理状況をネットワークで表現すると図 1 のようになる。

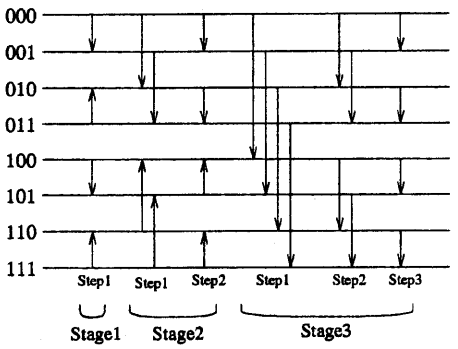

図 1　$n = 8$ のバイトニックソートのネットワーク

図 1 では、横線は各ノードでの処理の経過を示す。縦の矢印はデータの比較を示し、矢印の方向側に大きい値が与えられ、半対側に小さい値が与えられる。また、ステップ (Step) とは並列処理が可能な単位で、ステージ (Stage) とは双調列作成の単位になり、ステージ i では長さ 2^{i+1} の双調列 (長さ 2^i の単調列とも考えられる) を作成することになる。

2.2　バイトニックソートのアルゴリズム

$n = 2^r$ のデータのソーティングを考える。そのとき、バイトニックソートによるネットワークの各ステージ、ステップは以下のように構成される。ここで、データは各ノードに置かれている。また、2 進数の最下位ビットの位置を 0 とする。

for $i = 1$ to r　(Stage i)
{
　for $j = 1$ to i　(Step j)
　{
　(1) ノードの組合せ決定
　　すべてのノードに対し、その番号の $i-j$ ビット目が異なるノードの各ペア (a, b) を考える。
　(2) 矢印の向き決定
　　(a, b) において、そのノード番号の i ビット目が 0 の場合は矢印を下向きにし、1 の場合は上向きにする。$i = r - 1$ のときはその上位のビットを 0 として考えすべての矢印を下向きにする。
　(3) データの比較
　　データの比較を行ない、矢印の方向側に大きいデータを、反対側に小さいデータを移動させる。
　}
}

3　ノード故障時のソーティング

3.1　故障の定義

ここであつかう故障は、ノードおよびそこからのリンクは使用不能である、と定義する。また、このような故障ノードを持つソーティングは、n 個のノードから故障ノードを除いた $n-1$ 個のノードのデータについてのソーティングと定義する。以下では、故障ノードが 1 個存在する場合のアルゴリズムについて述べる。

3.2　ソーティング可能な故障状態

ネットワーク型コンピュータに故障ノードが存在すれば、ほとんどの場合、バイトニックソートはそのままでは機能しない。しかし、故障のある状態でソーティングが可能になる場合もある。小林 [4] は、番号 0 のノードが故障している場合には、バイトニックソートの比較するデータの対を変更することなく、そのままの状態で、ソートが可能であることを示した。例えば、図 2 はデータ数 8 の場合の例であるが、ノード 0 が故障しているために、破線部分の処理が全て不可能になる。しかしながら、この場合、これをそのまま実行してもノード番号 1〜7 のデータはソートされる。図 3 は、ノー

ド 101 が故障した例で、この場合、これをそのまま実行してもソートできない。

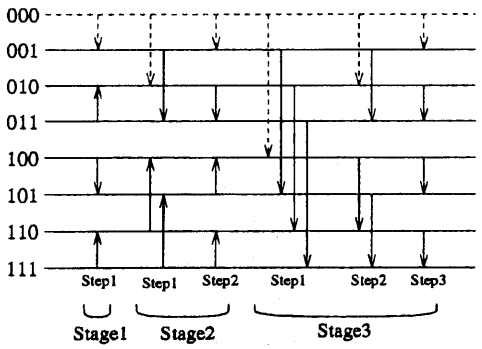

図 2　ノード 0 に故障を持つバイトニックソートのネットワーク

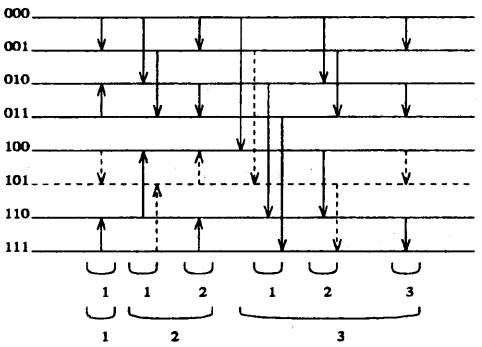

図 3　ノード 101 が故障したバイトニックソートのネットワーク

本論文のアルゴリズムは、基本的には、この性質を利用している。

3.3　故障ノードが 1 つのときのバイトニックソート

アルゴリズムの基本的な考えは、番号が 0 以外のノードが故障したときにも、ノード番号を変換して、故障ノードを 0 にしてしまうことである。そうすれば、上記の結果が利用でき、バイトニックソートのアルゴリズムでそのままソーティングが可能である。しかし、ここで問題が 2 つある。一つは、適切な再番号づけアルゴリズムを選ぶことである。もう一つは、でてきた結果が、変更した番号順にソートされているので、これを以下にして、元の番号順に戻すかである。したがって、以下で、ノード番号を再設定するアルゴリズムおよび再設定された番号の下でソートされたデータを、元の番号順に入れ換えるアルゴリズム（シフトアルゴリズムと呼ぶ）を示す。アルゴリズムの正当性の証明は次節で与える。

3.3.1　ノード番号の再設定

ネットワークのノード番号を変更する方法はノードが故障している場合にも応用が可能である。以下にノードが故障した場合のノードナンバーの変換によるアルゴリズムを示す。

1. 故障したノードの番号を x とする。ここで、$bin(x) = (x_{r-1}, \cdots, x_0)$, $x = x_{r-1}2^{r-1} + \cdots + x_1 2 + x_0$ と定義する。

2. すべてのノードに対し以下を行なう。そのノード番号を y としたとき、そのノードの新しい番号 z を $bin(z) = bin(x) \oplus bin(y)$ とする。ここで、$bin(x) \oplus bin(y)$ は各ビット毎の排他的論理和をとるものとする。

3. これによって、故障ノードの番号は 0 になる。

例えば、8 個のノードがある場合、101 のノードが故障しているとすると、各ノードの番号は以下のように再設定される。

000	→	101	001	→	100
010	→	111	011	→	110
100	→	001	101	→	000
110	→	011	111	→	010

この番号再設定アルゴリズムは以下の性質を持つ。

[補題 1]
任意の二つのノード x, y に対し、再設定前の番号のハミング距離と再設定後の番号のハミング距離は同じである。

証明
任意の二つのノード x, y を考え、それぞれのノード番号を x, y とする。そのとき、

$$bin(x) = (x_{r-1}, x_{r-2}, \cdots, x_0)$$

$$bin(y) = (y_{r-1}, y_{r-2}, \cdots, y_0)$$

ここで x,yのハミング距離を $l(x,y)$ とすると、

$$l(x,y) = (x_{r-1} \oplus y_{r-1}) + \cdots + (x_0 \oplus y_0)$$

と表せる。

次に x,yの変更後のノード番号を $new(x), new(y)$ とすると、$bin(new(x)), bin(new(y))$ は、故障ノードの番号を $(\alpha_{r-1}, \cdots, \alpha_0)$ としたとき、

$$bin\,(new(x)) = (x_{r-1} \oplus \alpha_{r-1}, \cdots, x_0 \oplus \alpha_0)$$

$$bin\,(new(x)) = (y_{r-1} \oplus \alpha_{r-1}, \cdots, y_0 \oplus \alpha_0)$$

と表せ、$l\,(new(x), new(y))$ は

$$\begin{aligned} & l\,(new(x), new(y)) \\ = \;& ((x_{r-1} \oplus \alpha_{r-1}) \oplus (y_{r-1} \oplus \alpha_{r-1})) \\ + \;& \cdots \quad + ((x_0 \oplus \alpha_0) \oplus (y_0 \oplus \alpha_0)) \\ = \;& (x_{r-1} \oplus y_{r-1}) + \cdots + (x_0 \oplus y_0) \end{aligned}$$

と表せる。

よって

$$l(x,y) = l\,(new(x), new(y))$$

となるので、任意のノード x,yについてノード番号変更前と変更後ではノード間のハミング距離は変わらない。

証明終

3.3.2 再設定したノード番号でのソーティング

ノード番号再設定後、障害ノードの番号は必ず0になる。したがって、ノード障害時のネットワーク型コンピュータ上でのバイトニックソートのアルゴリズムをノード番号の変更を行なってから実行すると、ノード 0 が故障した場合にはバイトニックソートは可能であるので、再設定した番号に基づいてデータはソートされる。

図 4 は番号再設定後のネットワークを示す。これの元は図 3 のネットワークであるが、番号を再設定した後、ネットワークは図 4 のようになる。

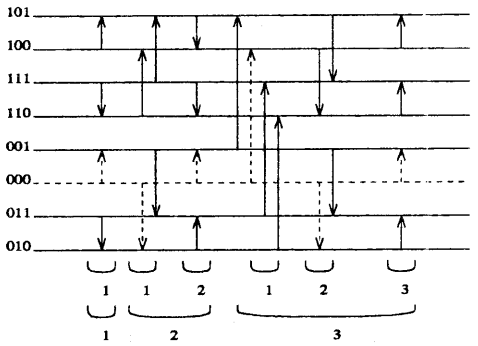

図 4　ノード番号変換後のバイトニックソートのネットワーク

3.3.3 シフトアルゴリズム

ノード番号を変更してソーティングされた列は元のノード番号を基準にするとソーティングできていない。そこで最終的にこの列を元のノード番号を基準にした状態でソーティングできているように各データをシフトする必要がある。ここでは、そのためのアルゴリズムを与える。

[シフトのアルゴリズム]

再設定番号によるバイトニックソートが完了した後、次のようなシフトステージを加える。

1. 初期状態のノード番号と再設定したノード番号を比べ、異なっているビットの中で最上位ビットの位置を i とする。
2. 初期状態のノード番号上で、バイトニックソートの最終ステージの $(\log n - i)$ ステップ以降を実行する。

以上で最終的に初期状態のノードナンバーを基準としたソーティングが完了する。

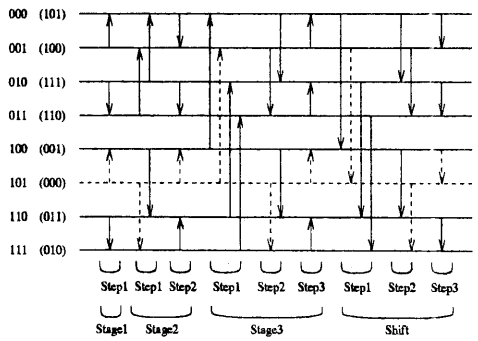

図 5　シフトステージを加えたバイトニックソートのネットワーク

4 証明

この節では、前節で与えた、再設定番号によるソーティング、シフトアルゴリズムにより最終的に最初のノード番号でデータがソートされることを示す。さらに、本アルゴリズムの計算量についても考察し、従来のものより効率が良いことも示す。証明は、再設定番号によるソートは、従来の結果より明らかであるから、後は、シフトアルゴリズムが正しく働くことを示せば良い。

まず、基本的な性質として、我々は以下の補題を得る。

[補題 2]

バイトニックソートの最終ステージ終了後は次のことが成り立つ。

故障ノードを x としたとき、

$bin(x) = (x_{n-1}, \cdots, x_0)$ とする。$x_j = 1$ なる $\forall j$、$\forall k \geq 0$ に対し

$\{a_{k2^{j+1}}, \cdots, a_{(k+1)2^{j+1}-1}\}$について

1.

$$\begin{aligned} & \min\{a_{k2^{j+1}}, \cdots, a_{k2^{j+1}+2^j-1}\} \\ \geq\ & \max\{a_{k2^{j+1}+2^j}, \cdots, a_{(k+1)2^{j+1}-1}\} \end{aligned}$$

2. $\{a_{k2^{j+1}}, \cdots, a_{(k+1)2^{j+1}-1}\}$に故障ノードが含まれているとき

$$\begin{aligned} & \min\{a_{k2^{j+1}}, \cdots, a_{k2^{j+1}+2^j-1}\} \\ =\ & a_{k2^{j+1}+x \mod 2^j} \end{aligned}$$

また $x_j = 0$ なる $\forall j$、$\forall k \geq 0$ に対して、またはステップ $\log n - j$ 終了後には

3.

$$\begin{aligned} & \max\{a_{k2^{j+1}}, \cdots, a_{k2^{j+1}+2^j-1}\} \\ \leq\ & \min\{a_{k2^{j+1}+2^j}, \cdots, a_{(k+1)2^{j+1}-1}\} \end{aligned}$$

4.1 シフトアルゴリズムの正当性の証明

Stage r を最終ステージ、シフトステージのステップを l とすると、

$x_{r-l} = 1$ であるとき

$$\{a_{k2^{r-l+1}}, \cdots, a_{k2^{r-l+1}+2^{r-l}-1}\}$$

$$\{a_{k2^{r-l+1}+2^{r-l}}, \cdots, a_{(k+1)2^{r-l+1}-1}\}$$

でシフトを行なう。

このときに故障ノード x は $a_{x-2^{r-l}}$ とシフトを行なうことになっているが、実際に $a_{x-2^{r-l}}$ はシフトを行なえないのでステップ l 終了後には、$k2^{r-l+1} \sim k2^{r-l+1} + 2^{r-l} - 1$ のノードには

$\{a_{k2^{r-l+1}+2^{r-l}}, \cdots, a_{k2^{j+1}+x \mod 2^{r-l}}, \cdots, a_{(k+1)2^{r-l+1}-1}\}$ $\cdots(A)$

が入り、$k2^{r-l+1} + 2^{r-l} \sim (k+1)2^{r-l+1} - 1$ のノードには

$$\{a_{k2^{r-l+1}}, \cdots, a_{k2^{r-l+1}+2^{r-l}-1}\}$$

$(a_{k2^{j+1}+x \mod 2^{r-l}}$は除く$)$ $\cdots(B)$

が入る。

ここで $r-l$ を j に置き換えると $(A), (B)$ は、

$\{a_{k2^{j+1}+2^j}, \cdots, a_{k2^{j+1}+x \mod 2^j}, \cdots, a_{(k+1)2^{j+1}-1}\}$

$\{a_{k2^{j+1}}, \cdots, a_{k2^{j+1}+2^j-1}\}$ $(a_{k2^{j+1}+x \mod 2^j}$は除く$)$

になる。

ここで補題 2 より、

$$\begin{aligned} & \max\{a_{k2^{j+1}+2^j}, \cdots, a_{k2^{j+1}+x \mod 2^j}, \\ & \cdots, a_{(k+1)2^{j+1}-1}\} \\ \leq\ & \min\{a_{k2^{j+1}}, \cdots, a_{k2^{j+1}+2^j-1}\} \\ & (a_{k2^{j+1}+x \mod 2^j}\text{は除く}) \end{aligned}$$

となるので、シフトのステップ l 終了後に $a_{k2^{j+1}+x \mod 2^j}$ のデータがシフトされずに残っても、(A) 側の値はすべて (B) 側の値を越えることはない。

またこの後も故障ノードを含むグループでは補題を満足する。

ここで (A) 側に残った $a_{k2^{j+1}+x \mod 2^j}$ のデータは、補題より $\max(A)$ となるので、次のステップでシフトする必要がある。

ここで $y = k2^{j+1} + x \mod 2^j$ とする。

a_y は次のステップ $(j = j-1)$ で $x_j = 1$ ならば

$$\min\{a_{k2^{j+1}}, \cdots, a_{k2^{j+1}+2^j-1}\}$$

と、$x_j = 0$ ならば

$$\min\left\{a_{k2^{j+1}+2^j},\cdots,a_{(k+1)2^{j+1}-1}\right\}$$

とシフトされる。

補題2の1、3より、x_jは0、1のどちらでもa_yは正しい位置にシフトされる。またa_yの位置にシフトされたデータはそのグループ内の最大値となるので、同様にすればよい。

またシフト中に故障ノードの影響を受けない部分が存在する場合もある。そこでは$\forall j$をつなぐリンクのみを使ってシフトを行なえばよいので、シフトステージをそのまま行なうことで代用できる。

よって$x_j=1$なる$\forall j$に対してシフトを行なうことによって初期状態のノードナンバー上でソーティングが完了する。以上で証明を終る。

4.2 計算量

データ数が$n=2^r$個のとき、バイトニックソートのステップ数の合計Sは、

$$S=\frac{1}{2}\log n\,(\log n+1)$$

である。

一方、本手法ではバイトニックソートのステップ数にシフトステージのステップを加えたものである。シフトステージは最大でも$\log n$ステップであるから、本手法のステップ数の最大値は、

$$\begin{aligned} S &= \frac{1}{2}\log n\,(\log n+1)+\log n \\ &= \frac{1}{2}\log n\,(\log n+3) \end{aligned}$$

となる。

5 ネットワーク型コンピュータへの適用

本論文で提案した方法は、バイトニックソートのネットワーク構造と本質的に同一の構造をしている。したがって、ネットワーク型コンピュータ上に本手法を適用するとき、バイトニックソートと全く同じ考えで行なえる。ここでは、有弦環ネットワークとハイパーキューブについて見てみる。

5.1 有弦環ネットワーク

小林 [5] は、バイトニックソートに$O((\log n)^2)$の冗長ステップを加えたアルゴリズムを提案している（しかし、このアルゴリズムでは番号の再設定は必要ない)。さらに、アルゴリズムの適用例として、有弦環ネットワーク型コンピュータ上でノード障害がある場合のデータのルーティングの考察を行なっている。

これと同じ手法を用いることにより、小林 [5] よりも効率良くソーティングができる。

5.2 ハイパーキューブ

バイトニックソートのネットワークの各リンクは、ハイパーキューブの各リンクに対応する。補題1は再設定番号づけは、ハイパーキューブの構造を変えない番号づけであることを意味している。したがって、再設定番号によるバイトニックソート、およびシフトアルゴリズムのネットワークのリンクもまた、ハイパーキューブの各リンクに対応している。このことは、ハイパーキューブネットワーク型コンピュータでは、高々$\log n$ステップの増加だけで、本論文のアルゴリズムが実行可能であることを意味する。

6 むすび

本論文では故障を持つネットワーク型コンピュータ上でバイトニックソートを行なう一手法を示した。

今後は、本手法の他のアルゴリズムにおいての有効性や、更に効率的なアルゴリズム、また故障箇所が複数個ある場合などについて検討する必要があると考える。

参考文献

[1] Selim G. Aki, Prallel sorting algorithm, Academic Press Inc., 1985.

[2] F.Thomson Leighton, Intoroduction to parallel algorithms and architectures, Morgan Kaufmann, 1992.

[3] 小林 洋, 山本 博章, 山浦 弘夫, リンク障害時の有弦環結合コンピュータにおけるソーティング, 信学論 (D-I), Vol. J75-D-I, No.5, pp.280-287, May 1992.

[4] 小林 洋, 山本 博章, 山浦 弘夫, ステップ数を増やさずにできるノード障害時のネットワーク型コンピュータ上でのバイトニックソート, 信学論 (D-I), Vol.J75-D-I, No.10, pp.958-961, Oct. 1992.

[5] 小林 洋, 船木 英岳, 山本 博章, 山浦 弘夫, ノード障害時のネットワーク型コンピュータ上でのバイトニックソート, 信学論, Vol.J76-D-I, No.9, pp.465-472, Sep. 1993.

[6] 小林 洋, 船木 英岳, 山本 博章, 山浦 弘夫, ノード障害時のネットワーク型コンピュータ上でのバイトニックソートの改良法, 信学論, Vol.J77-D-I, No.3, pp.266-270, March 1994.

[7] J. Sun, E. Cerny and J. Gecsei, Faout Tolerance in a Class of Sorting Networks, IEEE Trans. on Computers, Vol.43, No.7, 1994.