「超並列コンピュータにおける障害時の処理方法の研究」

小林　洋

「超並列コンピュータにおける障害時の処理方法の研究」

１９９４年3月

小林　洋

研究成果の概要

数千から数万個のプロセッサをネットワーク結合して並列に動作させる超並列コンピュータ（Massively parallel computer）は、コンピュータの処理速度を向上させる方法として有望であり、ＶＬＳＩ技術等の進歩により実用となりつつある。

超並列コンピュータにおいては、ネットワークを構成するノードのプロセッサやリンクの一部が障害を起こした時に、いかに処理を続行させるかということが問題として上げられる。

本論文は、超並列コンピュータ上でノードやリンクの一部に障害がある場合に、コンピュータの処理のうちでも重要なソーティングを継続して行う方法についての研究の成果をまとめたものである。

本論文の構成は次のとおりである。

まず、１章で研究の概要を述べる。

２章では、プロセッサを有弦環（Chordal Ring）型に結合したネットワークにおいて並列処理向きのソーティングアルゴリズムであるバイトニック・ソートを使って並列にソーティングを行う方法について述べる。

３章では、有弦環結合ネットワークを構成するリンクのうち任意の１本または２本が障害を起こした時に効率的に代替経路を取る手法をバイトニック・ソートを例に上げ示す。

４章では、ネットワークのノードに当たるプロセッサが障害を起こした場合に、障害ノードの影響を除きバイトニック・ソートを行う手法について述べる。

本研究は、著者が防衛大学校理工学研究科所属時に行った２章の研究を基に、本学において３ ～ ４章の研究に発展させたものである。

目 次

1 序論

コンピュータの処理能力を向上させるために、数千個から数万個の小型プロセッサをネットワーク状に結合して並列処理を行う、いわゆる超並列方式が研究されており、商用機として実現されているものもある。[(1)~(4)]

このようなコンピュータは、超並列コンピュータと呼ばれている。

この技術的背景には、近年のＶＬＳＩ技術や光通信技術等の進歩が上げられる。[(5)]

コンピュータで行う処理のうち、データの照合や探索のためにデータを一定順序に配列し直すソーティングは全処理時間のうちかなりの割合を占める重要な処理である。

超並列コンピュータ上でのソーティングについては、いくつかの結合方式について研究がなされている。[(7)~(12)] このうち正方格子（Mesh [(7)]）結合においては、データを各々１個ずつもつ n 個のプロセッサからなる超並列コンピュータ上でバイトニックソート [(13)] という手法により $O(\sqrt{n})$ の時間で処理できることが知られている。[(7)(8)]

ネットワークの各ノード当たりのリンク数 k は、正方格子結合においては（周辺部に折り返しがあるものとすれば）$k=4$ となる。リンク数 k をコストに関わるパラメータと考えると、k は少ない方が好ましいことになる。

そこで本論文では、k をさらに１つ減らした $k=3$ の有弦環（Chordal Ring [(6)]）結合をとりあげ、ノード数 n 、データ数 n の有弦環結合ネットワークにおいては、ソーティングの処理時間は $O(\sqrt{n}\log n)$ となることを示す。

ところで、ネットワーク型超並列コンピュータにおいては、障害時の処理方法が問題として上げられる。

ネットワークを構成するノードやリンクの一部に障害が発生した時、障害箇所の影響を除き処理を継続する手法が必要となる。

リンク障害時には、代替経路を用いることにより処理の継続が可能となる。

そこで本論文では２番目に、リンク障害時の有弦環結合ネットワーク型超並列コンピュータ上での代替経路のたどり方について、リンク障害が１本の場合と２本の場合についてバイトニック・ソートを例に上げ示す。

また、有弦環結合ネットワークにおいては、リンクの代替経路長は常に５となり、障害リンクが１ヶ所の時には全経路長の増加は $O((\log n)^2)$ 程度にとどまることを示す。

一方、ノード障害時においては、単に代替経路を求めるだけでなく、障害ノードで行うはずだった処理の影響についても考慮しなければならない。

そこで本論文では３番目に、ノード障害時の超並列コンピュータでのバイトニック・ソートの一般的手法を示し、有弦環結合ネットワークへの適用方法を示す。

ノード障害時には、あらかじめ処理に冗長性を持たせ、障害ノードの位置によりアルゴリズムの一部を変更することによりソーティングが可能となることを示す。

(注)
障害（failure）、誤り（error）、フォールト［又は故障］（fault）という3つの用語については、次のように区別して使われている。[1]

システムの構成要素の「フォールト」の結果、構成要素の出力の「誤り」状態が発生し、その結果ユーザから見たサービスの異常「障害」が起きる。但し、これらは相対的なものである。

本論文では、「ソーティング」というサービスに対する異常という観点からノードやリンクの「障害」という用語を使った。

[1] J,C.Laprie :" *Dependable computing and tolerance : concepts and terminology* ", Dig, Pap., FTCS-15,pp.2-11(June 1985).

2　有弦環結合ネットワーク型超並列コンピュータにおけるソーティング

ネットワーク型コンピュータによるソーティングとは、ネットワークの各ノードにあたるプロセッサが分散して持つデータを、プロセッサに一連に割りつけてある順序に従って並べ直すことである。その並列処理の手法については、いくつかの研究が行われている。[7]~[12]

ところで、多数のプロセッサを接続するには、結合が規則的で、各ノードのリンク数 k が少なく、かつ、最大ノード間距離 D が短いことが望ましい。

一般に k と D とはトレードオフの関係にある。

ノード数が n の場合、$k=2$ の場合環結合となり $D=n/2$ となってしまうが、$k=3$ においては、環結合の各ノードに弦を1本ずつ加え、弦を有弦環（ Chordal Ring ）と呼ばれる形状に結合することによって $D=O(\sqrt{n})$ となることが知られている。[6]

本章では、この性質を用いて、プロセッサを有弦環結合型にネットワーク接続し、その上で並列ソーティングを行う手法について述べることにする。

以下、**2.1**では有弦環結合について述べ、**2.2**では各ノードにデータが1個ずつある場合のソーティング、**2.3**では各ノードに複数個のデータがある場合のソーティングの方法について述べる。

2.1　有弦環結合

本論文でとり上げるリンク数 $k=3$ の有弦環結合とは、次のような結合をいう。図2．1のように環結合において環に沿って一方向にノードにインデックスをつけ、一定の番号おきに2つのノードを弦（ chord ）で結合する。

このとき、弦は奇数番目ノードとそこから時計回り方向にある偶数番目のノードとを接続する。

n 個のノードからなる有弦環結合において、ある弦の両端のノード間を弧 (arc) に沿って進んだときの距離を弦の長さ (chord length) と呼び w で表す。

このとき w は、$w \leq n/2$ で、$w=n/2$ を除き奇数となる。最大ノード間距離 d が最小になる弦の長さは $w \approx \sqrt{n}$ であり、このとき $d \approx \sqrt{n}$ となることが知られている。[6]

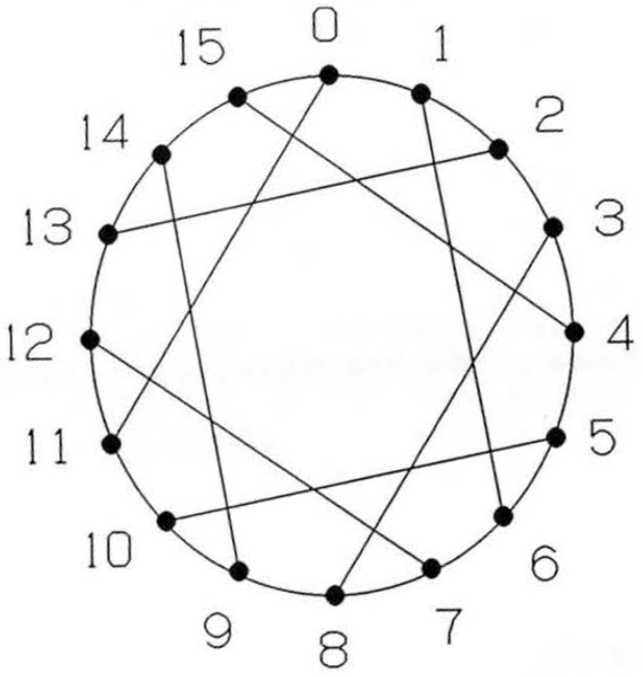


図2．1　有弦環結合

Fig.2.1 Chordal ring connection.

2.2 各ノードがデータを1個ずつ持つ場合のソーティング

2.2.1 バイトニック・ソート[13][14]

有弦環結合コンピュータ上でソーティングを行うには、バイトニック・ソートという手法が適している。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。

双調列とは、昇順と降順の2つの単調列（ monotonic sequence ）からなる順序列であり、順序列（ $a_1, a_2, \cdots, a_N$ ）が双調列であるとき $a_1 \leq a_2 \leq \cdots \leq a_i \geq a_{i+1} \geq \cdots \geq a_N (1 \leq i \leq N)$ なる関係が成り立っている。又、双調列を巡回シフトしてできる系列も双調列という。双調列（ $a_1, a_2, \cdots, a_N$ ）において次のような2つの順序列を作る。

$$(\min(a_1, a_{N/2+1}), \min(a_2, a_{N/2+2}), \cdots, \min(a_{N/2}, a_N))$$

$$(\max(a_1, a_{N/2+1}), \max(a_2, a_{N/2+2}), \cdots, \max(a_{N/2}, a_N))$$

すると、min をとった順序列の各値は、max をとった順序列の各値を超えることがない。又、各々の順序列は双調列になっている。

この性質を再帰的に用いながら、次のようにしてソーティングを行う。

（1）2^i 個のデータの単調列を2組ずつ組み合わせて、2^{i+1} 個のデータの双調列を作る。

（2）2^{i+1} 個のデータの双調列から 2^{i+1} 個のデータの単調列を作る。この際、上記の双調列の性質を使用して、2^{i+1} 個のデータの双調列から2組の 2^i 個のデータの双調列、さらにそこから 2^2 組の 2^{i-1} 個のデータの双調列、…と再帰的に繰り返し、2^{i+1} 個のデータの単調列を生成する。

（1）、（2）を繰り返すことにより、データ数 $N = 2^p$ の時、双調列の長さを $2^2, 2^3, \cdots, 2^p$ として行き、最後に長さ 2^p の単調列を作成することによってソーティングを行う。[2]

データ数 $N = 16$ のときの処理状況は、図2．2のように表すことができる。[14,p237]

[2]証明は付録1参照

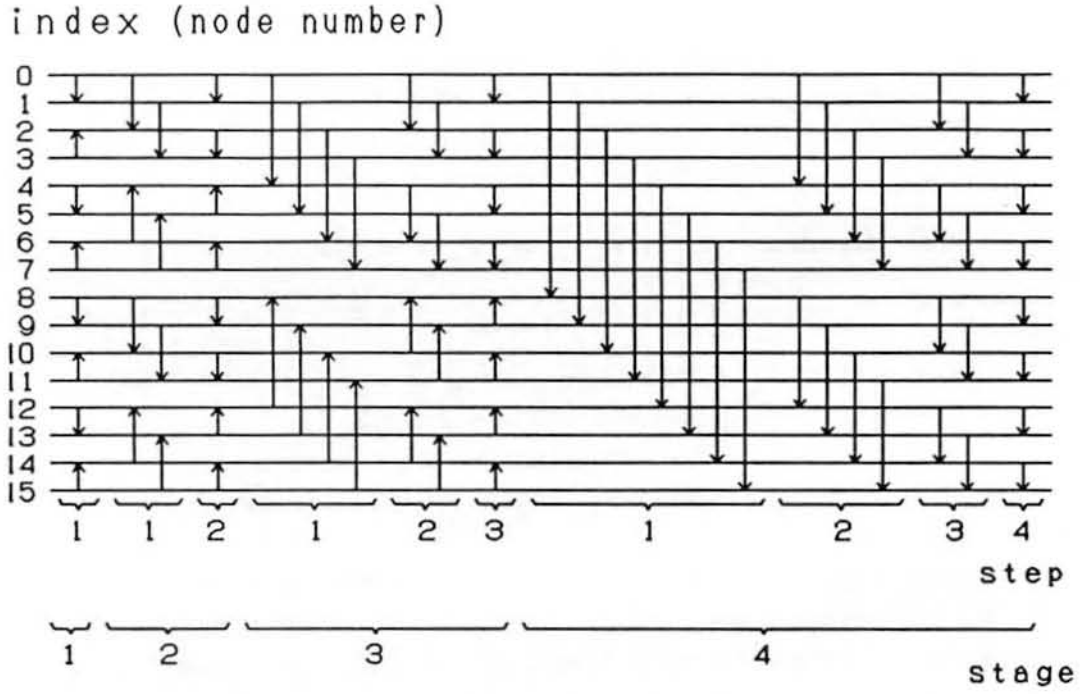


図2．2　バイトニック・ソートにおける処理状況　　$n = 16$

Fig.2.2 A process of bitonic sort.

この図においてデータは並列に左から入り右に出る。

矢印は、2つのデータの比較を表し、比較の結果矢印の方向側に大きい値が、反対側に小さい値が与えられることを示す。ここでステップ（step）とは並列処理が可能な単位である。ステージ（stage）とは双調列生成の単位でデータ数 $N = 2^p$ のとき i ステージにおいて長さ 2^i の単調列の対が、つまり長さ 2^{i+1} の双調列が $2^{p-(i+1)}$ 組でき、最終ステージで長さ 2^p の単調列ができることを示している。

バイトニック・ソートを有弦環結合ネットワーク型コンピュータで行うためには、図2．2を図2．1上へマッピングする必要があり、このマッピングの手法が、この章で述べる主要な点である。

2.2.2 ノードの構成と機能

各ノードのプロセッサは、経路レジスタ R_r（Routing register）と記憶レジスタ R_s（Strage register）の2つのレジスタを持ち、スワッピング機能があるものとする。R_r は3つのリンクに接続されている。[8]

バイトニック・ソートでは、半数のノードから他の半数のノードへデータを送り、そこで比較を行い、半数のデータを空になったノードに送り返すという動作を繰り返す。初めにデータを送る側のノードを送信ノード、データを受ける側のノードを受信ノードと呼ぶことにする。データは、初期状態では R_r にあり、送信ノードでは R_r から3つのポートのうちの1つを通してデータを送り出す。受信ノードでは比較すべきデータが到着するまで他のデータの流れを妨げないようにするため、R_s にデータを待避させておく。データの比較は、受信ノードの R_s と R_r との間で行われ、比較後送り返すデータを R_r に、自分のノードに保持しておくデータを R_s に入れ、R_r からデータを送り返す。各ノード間は、双方向通信が可能なものとする。

2.2.3 ノードの順番づけ

ノードにインデックスをつける方法について次に述べる。[7] ソーティングの処理時間をできるだけ短くするためには、頻繁にデータの比較が行われる2つのノード間ほど、物理的に短い距離になるように順番づけを行う必要がある。インデックスの値の差をインデックス距離（index distance）と呼ぶことにする。図2．2からも解るように、インデックス距離が短いほど頻繁にデータの比較が行われるので、図2・1のように時計回り方向に順番にインデックスをつけることにする。

2.2.4 弦の長さ

有弦環結合の性質より、弦の長さ w は $\sqrt{n}$ に近い奇数の値とする。[3]但し、$n=4$ のときは $w=2$、$n=8$ のときは $w=4$ である。又、最適な w の値は唯一つとは限らない。

2.2.5 制御手順

a．概要

制御手順の概要を次に示す。

（1）送信ノードと受信ノードを定める。比較するデータの入っている2つのノードのうち、インデックスの値の小さい方を送信ノード、大きい方を受信ノードとする。データを送信ノードから受信ノードへ送る時には有弦環上を時計回り方向に送り、受信ノードから送信ノードへ戻す時には反時計回り方向にデータを送る。

（2）受信ノードにおいては、データをレジスタ R_r から R_s に移動させる。

（3）送信ノードから受信ノードまでの最短経路に沿ってデータが動くように各ノードを制御する。

（4）受信ノードのレジスタ R_r に比較すべきデータが到着したら、R_r と R_s との間で比較を行う。

（5）比較後どちらのデータを戻すかを定め、自分で保持するデータを R_s に、戻すデータを R_r に移す。

（6）送信ノードにデータを戻す。データが戻った時点で1ステップが完了する。

b．送・受信ノードと距離の指定

送・受信ノードの指定及びデータの移動距離の指定は以下のように行う。ノード数は $n=2^r$ とする。

[手順1（送・受信ノードの指定）]　ノードのインデックスを y とすると、i ステージの j ステップで $\lfloor y/2^{i-j} \rfloor$ が偶数のノードは送信ノードで、奇数ノードは受信ノードである。但し、$0 \leq y \leq n-1$、$i=1,2,\cdots,r, j=1,2,\cdots,i$ であり、$\lfloor y \rfloor$ は y 以下の最大整数を表す。

[手順2（移動距離の指定）]　送信ノードでは、データをインデックス距離 $x=2^{i-j}$ だけ動かす。受信ノードでは、データをレジスタ R_r から R_s に移す。

C．経路の指定 (6)

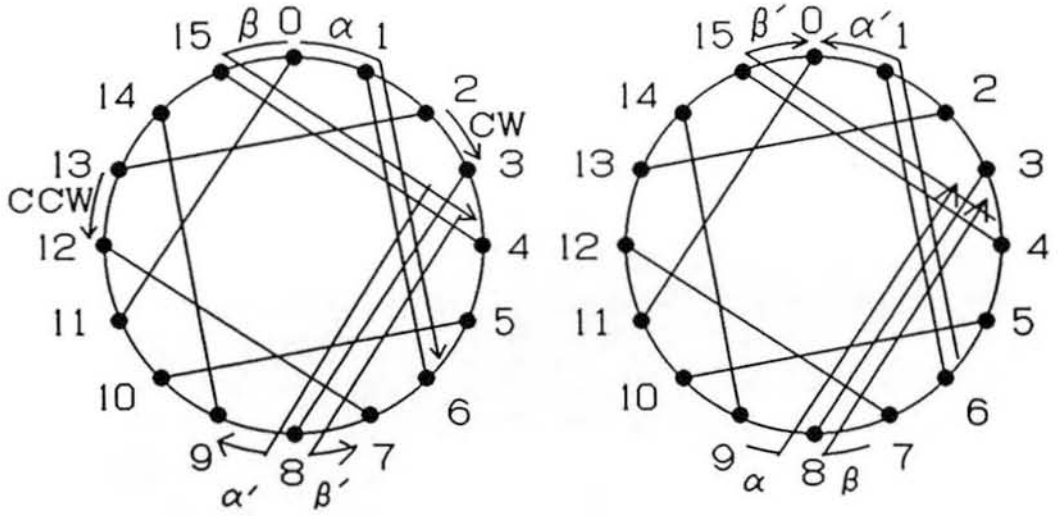
2つのノード間の最短経路は、1つとは限らない。例えば、図2．1においてノード0からノード8へ行く最短経路としては、（1）$0 \to 15 \to 4 \to 3 \to 8$、（2）$0 \to 1 \to 6 \to 7 \to 8$、（3）$0 \to 1 \to 2 \to 3 \to 8$ 等がある。

---

[3]厳密にいうと、$w \doteqdot \sqrt{n}$ は最適ではない。
（付録2参照）

弦を経路に使う場合には、図２．３のような基本的経路を考える。時計回り方向（ＣＷ方向）では、偶数番目ノードから弧上を１つ進み次に弦上を１つ進む経路をα 経路、弧上を１つ戻り次に弦上を１つ進む経路を β 経路と呼び[4]、奇数番目ノードから弦上を１つ進み次に弧上を１つ進む経路をα' 経路、弦上を１つ進み次に弧上を１つ戻る経路をβ' 経路と呼ぶことにする。反時計回り方向（ＣＣＷ方向）についても同様な経路を考える。

[4]文献（６）の α-move 、β-move のことをここではα 経路、β 経路とした。



(a) Clockwise routes　　(b) Counter clockwise routes

図2．3　弦を使う場合の基本的経路

Fig.2.3 Basic routes by using chords.

時計回り方向の経路に関して次の3つの関数 $C(x)$ 、$P(x)$ 及び $Q(x)$ を考える。

$C(x)$: ($\alpha(\alpha')$ 経路の回数) + ($\beta(\beta')$ 経路の回数)

$P(x)$: ($\alpha(\alpha')$ 経路の回数) + (時計回り方向の弧上の経路の追加分)

$Q(x)$: ($\beta(\beta')$ 経路の回数) + (反時計回り方向の弧上の経路の追加分)

$C(x)$ 、$P(x)$ 、$Q(x)$ は次の手順によって与えられる。

[手順3 (経路の指定)] 送信ノードを0番目、受信ノードを x 番目とすると、バイトニック・ソートにおいて x の値は $x = 1$ を除き偶数であり、$\triangle x = x \ mod \ (w+1)$ とすると、

(1) $\lceil \frac{x}{w+1} \rceil \geq \frac{w-1}{2}$ のとき、

(a) $0 \leq \triangle x \leq 1$ ならば、

$$C(x) = \lfloor \frac{x}{w+1} \rfloor, P(x) = \lfloor \frac{x}{w+1} \rfloor + \triangle x, Q(x) = 0$$

(b) $2 \leq \triangle x \leq w$ ならば、

$$C(x) = \lceil \frac{x}{w+1} \rceil, P(x) = \lceil \frac{x}{w+1} \rceil - \lceil \frac{w-\triangle x}{2} \rceil$$ [5]

,

$$Q(x) = \lceil \frac{w-\triangle x}{2} \rceil \quad (x: \text{偶数})$$

(2) $\lceil \frac{x}{w+1} \rceil \leq \frac{w-3}{2}$ のとき、

(a) $0 \leq \triangle x \leq \frac{w+1}{2} - \lceil \frac{x}{w+1} \rceil$ ならば、

(1) - (a) に同じ

(b) $\frac{w+3}{2} - \lceil \frac{x}{w+1} \rceil \leq \triangle x \leq w - 2\lceil \frac{x}{w+1} \rceil$ ならば、

$$C(x) = \lceil \frac{x}{w+1} \rceil, P(x) = 0,$$

$$Q(x) = w + 1 - \lceil \frac{x}{w+1} \rceil - \triangle x$$

(c) $w - 2\lceil \frac{x}{w+1} \rceil + 1 \leq \triangle x \leq w$ ならば

(1) - (b) に同じ

となる。但し、$\lceil x \rceil$ は x 以上の最小整数を表し、$\lfloor x \rfloor$ は x 以下の最大整数を表す。

従って、手順2で求まる x の値を手順3に代入すれば、$C(x)$ 、$P(x)$ 及び $Q(x)$ の値が求まる。

反時計回り方向についても同様に求めることができる。

d. 最短経路の決定とデータの転送

手順3で求まる $C(x)$ 、$P(x)$ 及び $Q(x)$ の値から $A(x)$: $\alpha(\alpha')$ 経路の回数、$B(x)$: $\beta(\beta')$ 経路の回数、$V(x)$: 時計回り方向の弧上の移動の追加分、$W(x)$:反時計回り方向の弧上の移動の追加分を次のように決定する。

[手順4 (最短経路の決定)]

(1) $C(x) = 0$ のとき、

$$A(x) = 0, B(x) = 0,$$

$$V(x) = P(x), W(x) = Q(x)$$

(2) $C(x) \neq 0$ のとき、

[5]文献 (6) では、$P(x) = \lceil \frac{x}{w+1} \rceil$ であるが、ミスと考えられる。

（ａ）$C(x) \geq P(x)$ ならば、

$$A(x) = P(x), B(x) = C(x) - P(x),$$
$$V(x) = 0, W(x) = P(x) + Q(x) - C(x)$$

（ｂ）$C(x) < P(x)$ ならば、

$$A(x) = C(x) - Q(x), B(x) = Q(x),$$
$$V(x) = P(x) + Q(x) - C(x), W(x) = 0$$

となる。

手順4の結果をもとに、時計回り方向のデータの流れの制御を次の手順で行う。

[手順5（データの転送１）]

（１）$A(x)$ の回数だけ $\alpha(\alpha')$ 経路にデータを送る。

（２）$B(x)$ の回数だけ $\beta(\beta')$ 経路にデータを送る。

（３）$V(x) = 0$ のときは、$W(x)$ の回数だけ弧上を反時計回り方向にデータを送る。

$W(x) = 0$ のときは、$V(x)$ の回数だけ弧上を時計回り方向にデータを送る。

手順5の α 及び α' 経路並びに β 及び β' 経路については、次のように各ノードを制御する。

[手順6（データの転送２）]

（１）α 及び α' 経路の制御　　偶数番目ノードでは、レジスタ R_r にあるデータを、弧上のインデックスが１つ増大する方向にあるノードに送る。奇数番目ノードでは、R_r にあるデータを、弦上のインデックスが w だけ増大する方向にあるノードに送る。

この動作を各々２回ずつ行う。

（２）β 及び β' 経路の制御　　偶数番目ノードでは、R_rにあるデータを、弧上のインデックスが１つ減少する方向にあるノードに送る。奇数番目ノードでは、R_r にあるデータを、弦上のインデックスが w だけ増大する方向にあるノードに送る。

この動作を各々２回ずつ行う。

例として、ノード数 $n = 16$ のときの β 及び β' 経路の制御によるノードの動作を図２．４に示す。各ノードとも矢印の方向へデータの送信を２回ずつ行う。

反時計回り方向についても同様にして制御が行われる。

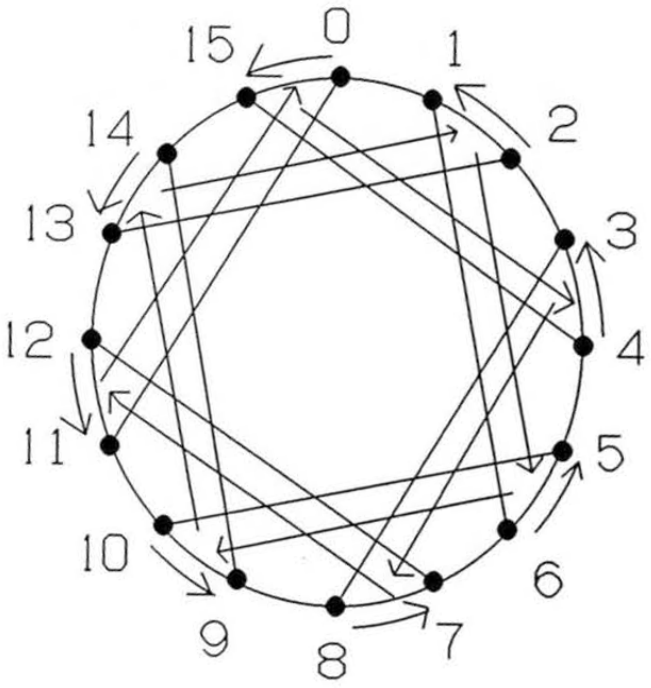


図2．4　β 及び β' 経路の制御

Fig.2.4 β and β' routes control.

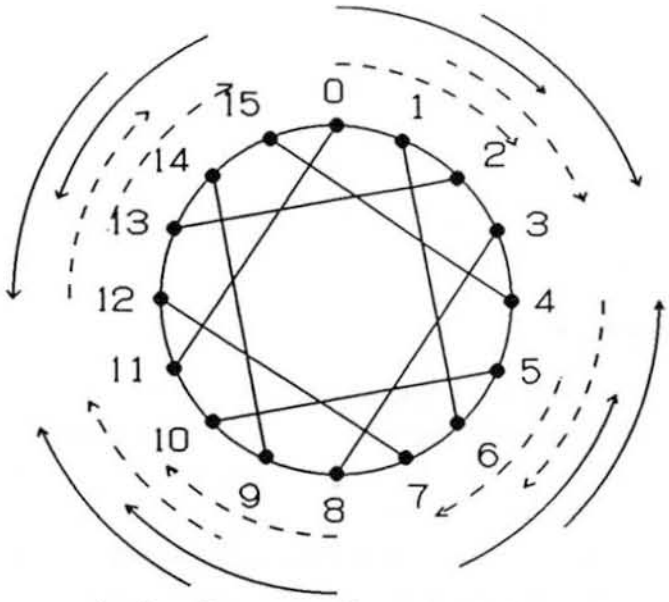
e．比較後の処理

受信ノードで比較後、送信ノードに戻すデータは次のように定める。

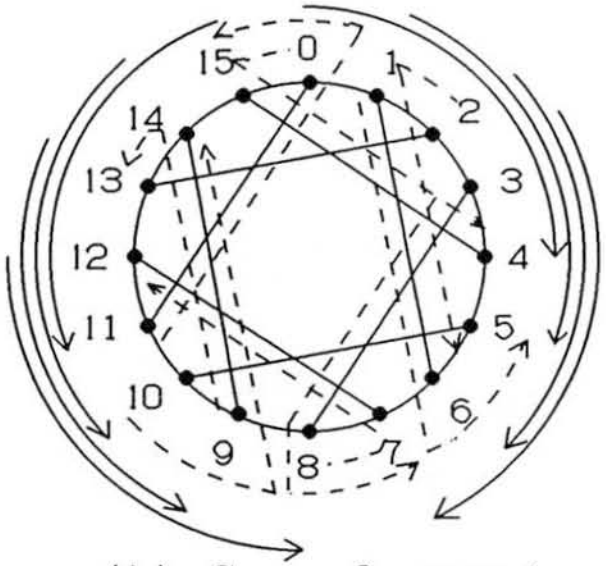
[手順7（比較後の処理）]　ノードのインデックスを y とすると、受信ノードにおいて、$\lfloor y/2^j \rfloor$ が偶数の場合には値の小さい方のデータを戻し、奇数の場合には値の大きい方のデータを戻す。もう一方のデータは、受信ノードで保持しておく。

図2．5に、$n = 16$ のときのステージ2、ステップ1とステージ3、ステップ1での送信ノードから受信ノードへのデータの流れ（破線）とノード間のデータ交換後のデータの大小関係（実線）を示す。

データの大小関係を表す実線の矢印は、図2．2に対応する。



(a) Stage2, step1



(b) Stage3, step1

----→ : Data streams from transmitting nodes to receiving nodes

——→ : Magnitude of data after data exchange

図2.5 データの流れ

Fig.2.5 Data streams.

2.2.6 処理時間の評価

ネットワーク型コンピュータで処理時間を評価するためには、データの経路通過時間と比較時間を求める必要がある。[7] まず、経路通過時間を求めるために、並列的に動作したときの送信ノードから受信ノードまでの物理的距離の合計（以後全距離と呼ぶ）を求める。

例えば、$n = 16, w = 5$ の場合、処理が行われるノード間のインデックス距離 x_i 、物理的距離 l_i 及びその距離におけるデータ転送の頻度 f_i（同じノード間のデータ転送の回数）について図２．１及び図２．２より調べると表２．１のようになる。

表2．1

表2．1　距離と頻度 $(n = 16, w = 5)$

i	インデックス距離 x_i	物理的距離 l_i	転送頻度 f_i	弦の使用
0	2^0	1	4	使用せず
1	2^1	2	3	
2	2^2	2	2	使用する
3	2^3	4	1	

全距離 L は、次のように与えられる。

$$L = \sum_{i=0}^{3} l_i \cdot f_i = 18$$

一般に、ノード数 $n = 2^r$ 、弦の長さ w のとき、インデックス距離 x_i だけ離れた2つのノード間の物理的最短距離は、x_i が w に比べて大きく偶数の場合、およそ $2\lceil x_i/(w+1) \rceil$ となる。[6] 但し、$x_i = 2^i \leq (w+1)/2$ のうちは、つまりインデックス距離がおおよそ弦の長さの半分より小さい場合には弧に沿った方が近く、物理的最短距離は x_i となる。この関係を表2．2に示す。

表2．2

表2．2 距離と頻度 ($n = 2^r, w \approx \sqrt{n}$)

i	インデックス距離 x_i	物理的距離 l_i	転送頻度 f_i	弦の使用
0	2^0	2^0	r	
1	2^1	2^1	$r-1$	
2	2^2	2^2	$r-2$	使用せず
⋮	⋮	⋮	⋮	
i	2^i	2^i	$r-i$	
$i+1$	2^{i+1}	$2\left\lceil\frac{2^{i+1}}{w+1}\right\rceil$	$r-(i+1)$	
⋮	⋮	⋮	⋮	
$r-2$	2^{r-2}	$2\left\lceil\frac{2^{r-2}}{w+1}\right\rceil$	2	使用する
$r-1$	2^{r-1}	$2\left\lceil\frac{2^{r-1}}{w+1}\right\rceil$	1	

全距離 L は、次式により与えられる。

$2\lceil x_i/(w+1)\rceil \approx 2x_i/(w+1)$ より、

$$L=\sum_{i=0}^{r-1} l_i \cdot f_i \doteq 2^0 r+2^1(r-1)+\cdots+2^i(r-i)+(2\frac{2^{i+1}}{w+1}+1)(r-(i+1))+\cdots+(2\frac{2^{r-1}}{w+1}+1)\cdot 1$$

よって、

$$L \doteq 2^{i+1}(r-i+1)-r-2+\frac{1}{w+1}(2^{r+2}-2^{i+3}-2^{i+2}(r-i-1))+\frac{1}{2}((r-i)^2-(r-i))$$

$2^i \approx (w+1)/2, w+1 \approx \sqrt{n}$ より

$i \approx (\log n)/2-1$ 又、$n=2^r$ より $r=\log n$

よって、式（1）が求まる。

$$L \doteq \frac{1}{2}\sqrt{n}(\log n+12)-2\log n-6 \tag{1}$$

（対数は全て 2 を底とし、以下同様とする。）

弧又は弦上を単位距離（隣りあった 2 つのノード間の距離）データ転送するのに要する時間を t_r とすると、データの移動に必要な全時間 T_R（以後全経路時間と呼ぶ）は、往復必要なので、次式により与えられる。

$$T_R=(2L)t_r=(\sqrt{n}(\log n+12)-4\log n-12)t_r \tag{2}$$

又、比較に必要な全時間 T_c（以後全比較時間と呼ぶ）は、単位時間を t_c とすると、次式により与えられる

$$T_c=(\sum_{i=0}^{r-1} f_i)t_c=(\frac{1}{2}\log n(\log n+1))t_c \tag{3}$$

式（2）の主要項は $\sqrt{n}(\log n+12)$ であり、この値は n が 2^{12} に比べ小さいときには $O(\sqrt{n})$ 、となり、これは正方格子（Mesh）結合ネットワーク上での値[6]とほぼ同程度である。

2.3 各ノードがデータを複数個持つ場合のソーティング

各ノードのプロセッサが記憶部を持ち、その中にデータが複数個ある場合のソート法について述べる。

このような場合、従来考えられていた並列処理の方法としては、*merge-splitting* の方法 (7)(14) がある。これは各ノードに等分したデータを内部ソートとマージの組み合わせによりソーティングを行う方法である。しかし、この方法は余分な記憶領域を必要とする。

ここに示す方法は、バイトニック・ソートを使った方法で、余分な記憶領域は不要である。

[6]付録 3 参照

2.3.1 バイトニック・ソートのみを使う方法

データ数 $N = 2^p$, ノード数 $n = 2^r$ とすると、ノード1個あたりのデータ数は $M = N/n = 2^{p-r} = 2^s$ となる。

弦の長さ w は、$w \approx \sqrt{n}$ となる奇数の値とする。但し、$n = 4$ のとき $w = 2$、$n = 8$ のときは $w = 4$ である。

各ノードの記憶部のデータのある位置に、次のように一連のインデックスをつける。まず、基準となる0番目のノードの記憶部にある M 個のデータに0から $M-1$ までの番号を与え、次に時計回り方向のノードにあるデータに M から $2M-1$ までの番号を与え、以下同様に、時計回り方向に次々と番号をふっていく。

このように番号を与えたネットワークにおいて、バイトニック・ソートを実行すると、比較するデータが同一ノード内にある場合と、別のノードにある場合がある。(図2.2, 図2.6参照)

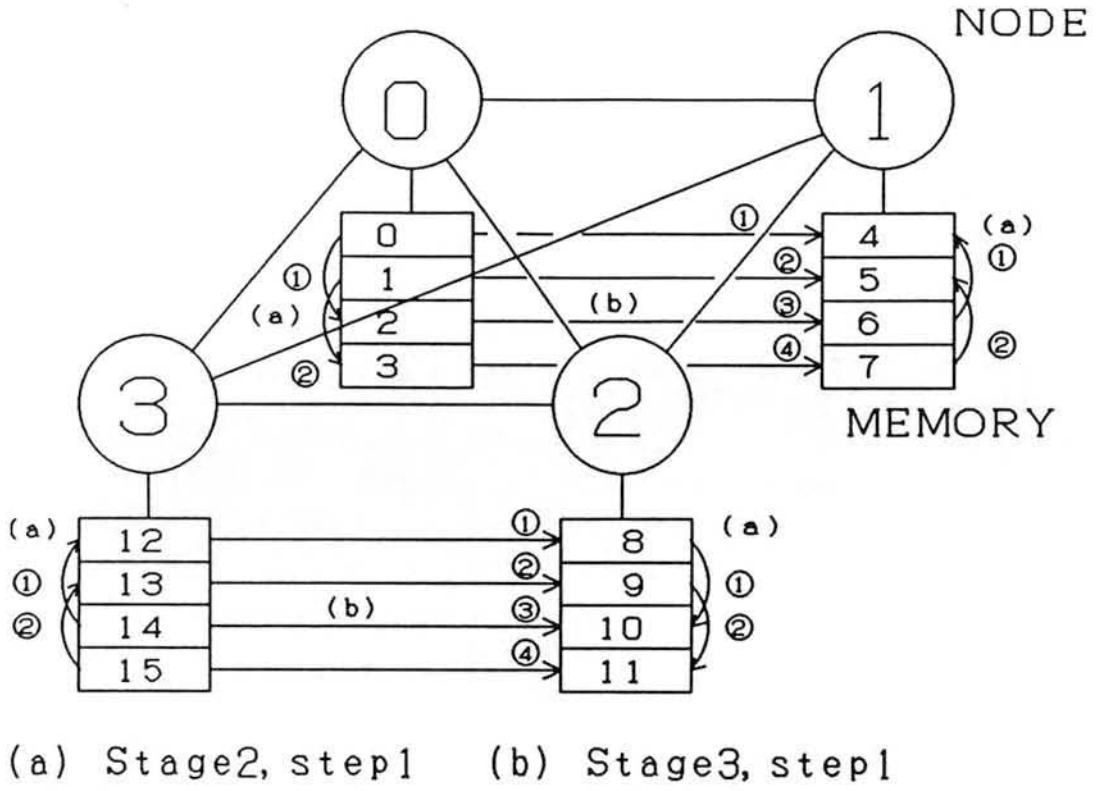


図2．6　各ノードにデータが複数個ある場合の処理状況 $N = 16, n = 4$

Fig.2.6 A process of multi-data node.

a．制御手順

（1）インデックス距離 $x < M$ の場合

比較はノード内で行える。従って1回の比較を行うためには、比較する2つのデータを記憶部からレジスタ R_r と R_s にロードし、R_r と R_s の間で比較を行い、R_r と R_s から記憶部にストアする必要がある。つまり、1回の比較に4回のアクセスが必要になる。（図2．7（a）参照）

（2）$x \geq M$ の場合

比較はノード間で行う。従って1回の比較を行うためには、比較する2つのデータが別のノードの記憶部にあり同時にアクセスが行えるため、2回のアクセスでよい。まず、比較するデータをそれぞれのノードの R_r にロードし、**2.2.5** の制御手順に従ってデータを動かす。送信ノードにデータが戻ったならば、記憶部にストアする。（図2．7（b）参照）

1つのノードが複数個のデータを持つ場合、各ステップは完全には並列処理ができなくなり、1ステップを実行するのに、$x < M$ のときには $M/2$ 倍、$x \geq M$ のときには M 倍のステップが必要になる。

例えば、$M = 4$ の場合には、図2．6のように、1ステップが $x < 4$ のときには2ステップに、$x \geq 4$ のときには4ステップになる。

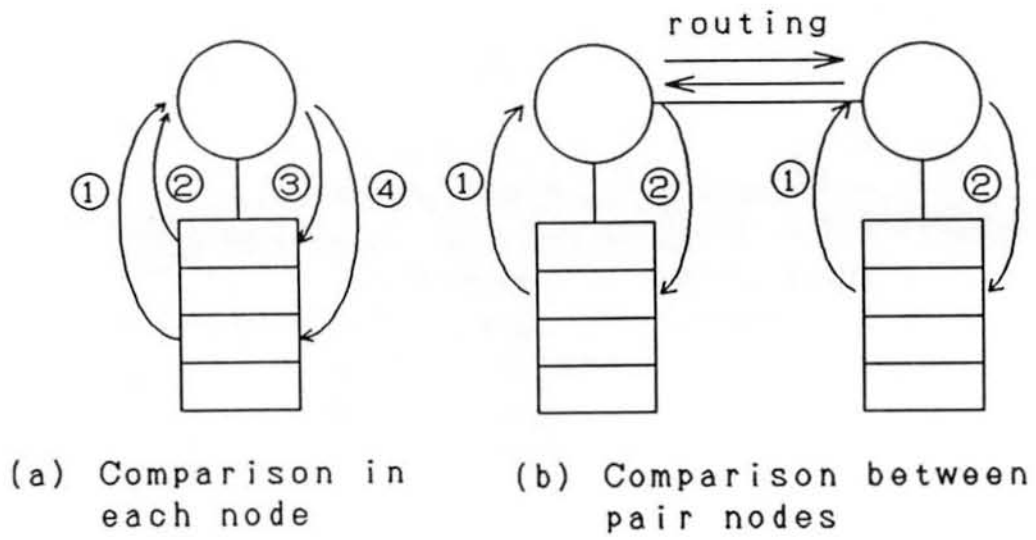


図2．7　アクセス回数

Fig.2.7 The number of memory access.

b．処理時間の評価

データ数 $N = 64$ 、ノード数 $n = 16$ 、ノード1個あたりのデータ数 $M = N/n = 4$ 、弦の長さ $w = 5$ の場合のインデックス距離 x_i、物理的距離 l_i 及び転送頻度 f_i 等について調べると表2．3のようになる。ここでアクセス距離 h_i とは、1回の比較に必要なメモリアクセスの回数で、ノード内比較では、$h_i = 2$ 、ノード間比較では、$h_i = 1$ となる。又、重み m_i とは、完全な並列処理に比べ何倍のステップが必要であるかを示し、ノード内比較では $m_i = M/2$ 、ノード間比較では $m_i = M$ である。

表2．3

表2．3 距離と頻度等 ($N = 64, n = 16, w = 5$)

i	インデックス距離 x_i	物理的距離 l_i	アクセス距離 h_i	転送頻度 f_i	重み m_i	弦の使用	比較データの位置
0	2^0		2	6	2		ノード内
1	2^1		2	5	2		
2	2^2	1	1	4	4	使用せず	
3	2^3	2	1	3	4		ノード間
4	2^4	2	1	2	4	使用する	
5	2^5	4	1	1	4		

ノード間の経路の全距離 L_N は、

$$L_N = \sum_{i=0}^{5} l_i \cdot f_i \cdot m_i = 72$$

となり、ノード間及びノード内比較のアクセスのための全距離 L_a は、次のようになる。

$$L_a = \sum_{i=0}^{5} h_i \cdot f_i \cdot m_i = 84$$

一般にデータ数 $N = 2^p$、ノード数 $n = 2^r$、ノード 1 個当りのデータ数 $M = N/n = 2^{p-r} = 2^s$ の場合、L_N は式（1）において M 倍した場合に相当するので次式により与えられる。

$$L_N = \sum_{i=0}^{p-1} l_i \cdot f_i \cdot m_i \doteq ML = \frac{N}{n}L = \frac{N}{n}(\frac{1}{2}\sqrt{n}(\log n + 12) - 2\log n - 6) \quad (4)$$

又、L_a について求めると、

$$L_a = \sum_{i=0}^{p-1} h_i \cdot f_i \cdot m_i = M\sum_{i=0}^{p-1} f_i = \frac{1}{2}\frac{N}{n}\log N(\log N + 1) \quad (5)$$

となる。

全経路時間 T_R は次式により与えられる。

$$T_R = (2L_N)t_r = (2\frac{N}{n}L)t_r = \frac{N}{n}(\sqrt{n}(\log n + 12) - 4\log n - 12)t_r \quad (6)$$

又、アクセスに必要な全時間 T_A は、単位アクセス時間を t_a とすると次式により与えられる。

$$T_A = (2L_a)t_a = \frac{N}{n}\log N(\log N + 1)t_a \quad (7)$$

全比較時間 T_c は、次式により与えられる。

i が $0 \leq i \leq s-1$ のときはプロセッサ内交換で $m_i = M/2$ 、$s \leq i \leq p-1$ のときはプロセッサ間交換で $m_i = M$ となるので、

$$\begin{aligned} T_c &= \left(\sum_{i=0}^{p-1} f_i \cdot m_i\right) t_c = \left(\frac{M}{2}\left(\sum_{i=0}^{s-1} f_i\right) + M\left(\sum_{i=s}^{p-1} f_i\right)\right) t_c \\ &= \frac{M}{4}(p(p+1) + (p-s)(p-s+1))t_c \end{aligned} \quad (8)$$

$s = p - r, p = \log N, r = \log n$ を式（8）に代入すれば次式が得られる。

$$T_c = \frac{1}{4}\frac{N}{n}(\log N(\log N + 1) + \log n(\log n + 1))t_c \quad (9)$$

2.3.2 ヒープ・ソートを組み合わせた方法

データ数 $N = 2^p$ 、ノード 1 個当たりのデータ数 $M = 2^s$ のとき、バイトニック・ソートの $s(= \log M)$ ステージでは $M(= 2^s)$ 個のデータが昇順と降順に交互に並び、長さ 2^{s+1} の双調列を $2^{p-(s+1)}$ 組作っている。

ところで **2.3.1** の方法では、s ステージまでは各ノード内で直列に処理を行っている。バイトニック・ソートは、直列処理では M 個のデータを並び終えるまでに $M \log M(\log M + 1)/4$ 回の比較が必要である。[13]

そこで、バイトニック・ソートの 1 ステージから s ステージまでの部分を比較回数が $M \log M$ のオーダであるヒープ・ソートに置き換える方法が考えられる。但しこの場合には、ノードごとにデータの順序を昇順と降順に交互に並べるか、或は全てのノードで、昇順（又は降順）に並べた後インデックスを付け換える必要がある。この方法は、$s+1$ ステージ以後をみれば、マージと考えることができる。なお、$s+1$ ステージ以後の M 個の部分を並べ変えるのに要する比較回数は、各ステージとも $M \log M/2$ 回であるので、$s+1$ ステージ以後のノード内の M 個の並べ換えはヒープ・ソートに置き換えないものとする。

ヒープ・ソートを組み合わせた方法の処理時間は、次のようにして求めることができる。

この場合、$s+1$ ステージ以後は、バイトニックソートのみを使った方法と同じであるので、インデックス距離 $x < M$ の各比較部分の頻度は、インデックス距離 $x = M$ の頻度と等しくなる。

従って、$s+1$ ステージ以後のアクセスに必要な全時間 T_{A1} は、次式により求まる。

$$\begin{aligned} T_{A1} &= (2\sum_{i=0}^{p-1} h_i \cdot f_i' \cdot m_i)t_a \\ &= 2 \times (2 \times s \times (p-s) \times \frac{M}{2} + ((p-s) + (p-(s+1)) + \cdots + 2 + 1) \times M)t_a \\ &= \frac{N}{n} \log n(2 \log N - \log n + 1)t_a \end{aligned} \tag{10}$$

s ステージ以前は、ヒープソートを使うのでアクセスに必要な全時間 T_{A2} は次式により求まる。(C：係数)

$$T_{A2} = (4CM \log M)t_a = 4C\frac{N}{n} \log \frac{N}{n} t_a \tag{11}$$

従って、アクセスに必要な全時間 T_A は、

$$T_A = T_{A1} + T_{A2} = \frac{N}{n}(2\log N(\log n + 2C) - \log n(\log n + 4C - 1))t_a \quad (12)$$

となる。

$s+1$ ステージ以後の全比較時間 T_{C1} と s ステージ以前の全比較時間 T_{C2} は、次式により求まる。

$$\begin{aligned} T_{C1} &= (\sum_{i=0}^{p-1} f_i' \cdot m_i)t_c \\ &= (s \times (p-s) \times \frac{M}{2} + ((p-s) + (p-(s+1)) + \cdots + 2 + 1) \times M)t_c \\ &= \frac{1}{2}\frac{N}{n}\log n(\log N + 1)t_c \end{aligned} \quad (13)$$

$$T_{C2} = (CM\log M)t_c = C\frac{N}{n}\log\frac{N}{n}t_c \quad (14)$$

従って、全比較時間 T_C は、

$$T_C = T_{c1} + T_{c2} = \frac{1}{2}\frac{N}{n}(\log N(\log n + 2C) - \log n(2C - 1))t_c \quad (15)$$

となる。

全経路時間 T_R は、バイトニックソートのみを使った方法と同じとなる。

3　リンク障害時の有弦環結合ネットワーク型超並列コンピュータにおけるソーティング

有弦環結合においては、ノード数が n の場合、最大ノード間距離 $D = O(\sqrt{n})$ となることが知られている。[6]

そこで、リンクに障害が起きた時には、弦をうまく用いれば、比較的短い代替経路をとり得ることが予想できる。

本章では、有弦環結合において処理を行う場合、リンクの１本又は２本が障害を起こしても、代替経路をとることにより処理を実行することが可能な手法について **2** で述べたバイトニック・ソートを例に上げ示す。

以下、**3.1** では代替経路を容易に取りえるようにするためのデータ・フロー向きのアーキテクチャを示す。**3.2** では、リンクに障害がある場合の代替経路のたどり方を示し、**3.3** ではその評価を行う。

3.1　アーキテクチャ

2に示した手法は、全ノードで同期を取りつつデータを流していた。

これは、各ステップにおいて、データが一定のパターンに従い動いたために可能だった。

本章では、全ノードで同期を取らずに済むデータ・フロー向きの手法を用いることにする。この手法は、**3.2** で示す代替経路をとる際に有用となる。

以下に、そのために必要なアーキテクチャについて述べる。

3.1.1　連結表

各ノードには、あらかじめ他のノードへの最短経路の方向を示す連結表を用意しておく。

ソーティング等の処理を行う際には、この表に従ってデータを流す。

連結表は、有弦環結合においては次のようにすれば容易に作成することができる。

（１）　ネットワークの０番目と１番目のノードについて、表３．１のような他の $n-1$ 個のノードへの経路についての連結表（以下、基本連結表）を **2.2.5** に示した方法に従い作る。

表3．1

表3．1　基本連結表

	出発ノード	
目的ノード	0	1
0	−	0
1	1	−
2	1	2
3	1	2
4	15	2
5	15	6
6	1	6
7	1	6
8	15	6
⋮	⋮	⋮
15	15	0

（２）（１）で求めた基本連結表を基に、全偶数番ノードと全奇数番ノードの連結表を次のように作る。

〔連結表の作成方法〕

ノード番号 j の目的ノード i への連結表の値を $V_{i,j}$ とすると、基本連結表の値 $V_{i,0}, V_{i,1}$ を基にして他の連結表は次のように求めることができる。

$$V_{i,j} = (V_{i-2,j-2} + 2) mod\ n$$

但し、$i = 0, 1, \cdots, n-1$

$j = 0, 1, \cdots, n-1$

このように、連結表が容易に作れる点が有弦環結合の特徴の一つである。

データには送信時、目的ノード番号を示すヘッダを持たせ、各ノードでそのヘッダの値に従い、３本のいずれのリンクに流すか決定する。

3.1.2 ノードの機能とデータの形式

ノードの構成及び機能は、次のようにする。

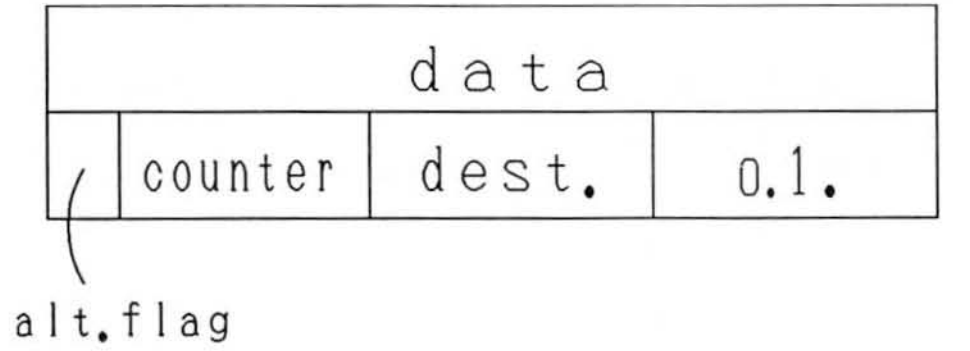
各ノードには、３つの全２重方式の通信路を結ぶバッファ、ＣＰＵ及び実行待テンプレートメモリがあるものとする。

各ノードは、前述の最短経路の方向を示す、$1 \times n$ の連結表、**3.2.3** で示すリンク障害時の初期駆動方向を示す単純なアルゴリズム及び簡単な代替連結表を持つ。

出発ノードにおいて目的地を書き込まれたデータは、各ノードの連結表に従ってネットワーク内を動く。

目的地のデータは実行待テンプレートメモリに入っており、出発ノードから比較の対象となるデータが到着すると比較が行われ、その後、アルゴリズムに従って大小いずれかのデータを出発ノードに送り返す。

データがノード間を動くときには、図３．１のトークンの形をとる。



alt.flag.:alternate route flag

counter:alternate route counter

dest.:destination

0.1.:operation level

図3.1　トークンの構造

Fig.3.1 A Structure of token.

トークンには、データの他に、目的地（目的ノード番号）、オペレーション・レベル（ステージとステップ判定用のタグ）、代替経路使用フラグ及び代替経路カウンタが必要となる。

オペレーション・レベルはステージとステップの通し番号であり、所定のデータ同士の比較を保障するものである。

代替経路使用フラグ及び代替経路カウンタの説明については **3.2** で示す。

3.2 リンク障害時の代替経路

有弦環結合の弦は、経路のバイパスとして使用する他に、効率的な代替経路として用いることができる。

環状結合においては、ノード間のリンクに障害が発生し、使えなくなった場合には、反対方向にデータを流すことによって、代替経路とすることが可能だが、ノード数 n の場合代替経路長 l は $l=n-1$ となってしまう。

これに比べ、有弦環結合においては弦を使用することにより、代替経路長を短くすることができる。

以下に、弦を用いた代替経路を用いる方法を示す。

有弦環結合の代替経路については、一手法が知られているが[6]、この手法を基にすると、データごとにいくつかの経路情報を記憶しておき、それを基に経由するノードごとに経路を計算し直す必要があり、処理の効率が悪い。

一方ここに示すのは、並列処理においてデータが効率的に代替経路を取り得る手法である。

3.2.1 弧の代替経路

弧の一ヶ所が切断された場合、弦を代替経路の一部として使用する。

CW 方向へデータが進む場合の代替経路について、以下に示す。なお、CCW方向についても同様に求めることができる。

（１）奇数番ノードの先の弧が断の場合

例えば、図３．２（ａ）においてノード１からノード２へデータを流そうとする場合、ノード１とノード２との間のリンクに障害があった時、ノード１からノード６の弦にデータを流し、その後、ノード２の方向へ弧を使って戻す１→６→５→４→３→２という弓形の経路がまず考えられる（以下、弓形経路）。

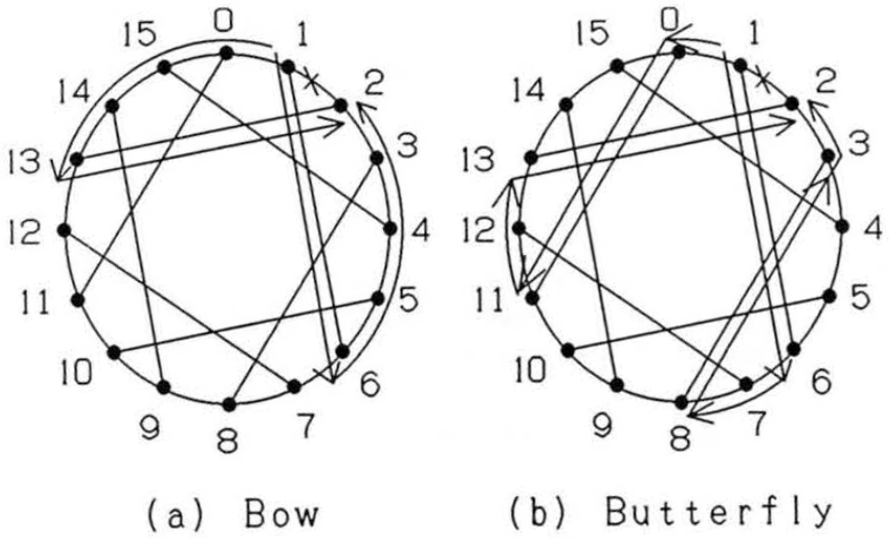


図3．2　弧の代替経路

Fig.3.2 Alternative routes of arc.

経路長 l は、

$$l = w$$

であり、この場合は、$w = 5$ であることから $l = 5$ となる。弓形経路は障害リンクに対して唯一ではなく、この他、1→0→15→14→13→2等、ノード1—2間を一部とする $\frac{1}{2}(w+1)$ 個の弓形経路が存在する。

しかしながら、弓形経路は一般的には、最短の代替経路とはならない。

有弦環結合においては、最大ノード間距離 D を小さくするために、$w \approx \sqrt{n}$ とすることから、$l \approx \sqrt{n}$ となり、l の値は n の平方根に比例してしまう。

一般的には弧の最短となる代替経路は、α 経路と β 経路を組み合わせた図3．2（b）に示す8の字形の経路である（以下、バタフライ形経路）。

つまり、この場合、1→6→7→8→3→2である。

バタフライ形経路においては、ノード数 n 、弦の長さ w によらず常に経路長は $l = 5$ となる。

バタフライ形経路は、1つの弧につき常に2個存在し、図3．2（b）の例では、上記の経路の他に1→0→11→12→13→2が存在する。

なお、弓形経路が最短の代替経路となるのは、$w = 3$ の時のみであり、$w = 5$ の時は、バタフライ形経路と経路長は同じで $l = 5$ となる。

（2）偶数番ノードの先の弧が断の場合

（1）の場合と同様、弓形経路と、バタフライ形経路とが存在する。

但し、弓形経路は $\frac{1}{2}(w-1)$ 個存在する。バタフライ形経路は（1）と同様2個存在する。

例えば、ノード0—1間断の時には、代替経路は0→15→4→5→6→1となり、バタフライ形経路では経路長さは常に $l = 5$ となる。

3.2.2 弦の代替経路

有弦環上のバイトニック・ソートでの経路設定時のアルゴリズムにおいては、**2.2.5** で示したように弦を用いる場合は、弧と組み合わせてα　経路、β 経路として取り扱ってきた。

しかしながら、データ・フローにおいては弦の代替経路を用いる場合には、α 経路や β 経路の代替経路を求めるのではなく弦自体の代替経路を求めた方が処理を行いやすい。

例えば、図3．3において、α 経路0→1→6に使われている弦1→6が障害を起こした場合、データは0から1に流れて1で初めて弦1→6の障害を知ることができると考えた方が良いだろう。そこから、弦1→6の代替経路として、1→0→15→4→5→6を取る。もし、ノード0で弦1→6の障害を知ることができるような機能があれば、ノード0から β 経路を使って経路長が上記のものより1短い0→15→4→5→6を取ることが可能であろうが、ハードウェアの構造が複雑になってしまう。

弦の代替経路については、弧の場合と同様、弓形経路とバタフライ形経路とが存在する。但し、一つの弦に対して弓形経路は1個のみ、バタフライ形経路は2個存在する。バタフライ形経路での経路長 l は、**3.2.2**と同様常に $l=5$ となる。

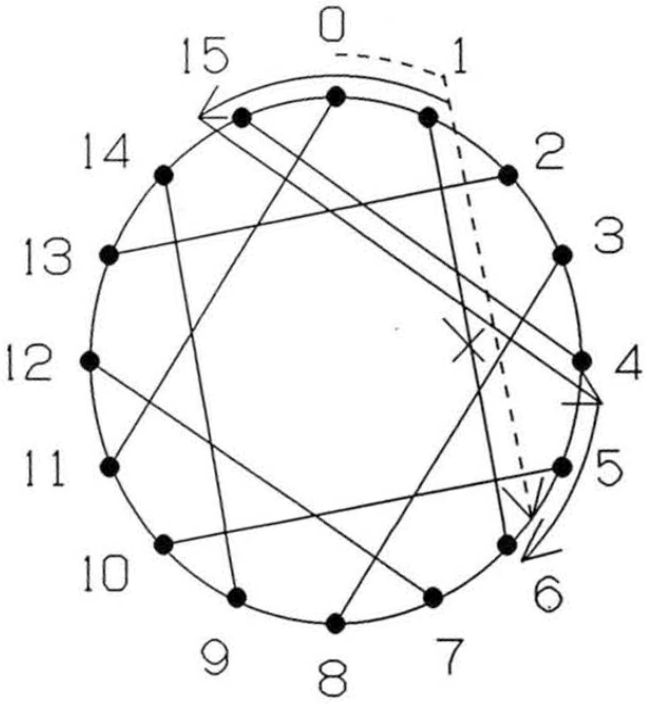


図3．3　弦の代替経路（バタフライ形経路）

Fig.3.3 Alternative routes of chord.(Butterfly)

3.2.3　代替経路のたどり方

3.2.1、**3.2.2**では、有弦環結合において、弧又は弦となるリンクの１本が障害を起こした時、その最短の代替経路はバタフライ形経路で各リンクごとに２本あることを示した。

代替経路をたどる手法としては、この内の１本を各ノードで用いることとし、各ノードに代替経路の方向を示す表３．２の代替連結表を持たせておく。

各ノードの代替経路の制御方法は次のとおり。

（図３．４参照）

表3．2

表3．2 代替連結表

区分	入力方向	出力方向
偶数番ノード	$\bar{S}$	$\bar{C}$
	S	$\bar{S}$
	$\bar{C}$	S
奇数番ノード	$\bar{S}$	S
	S	C
	C	$\bar{S}$

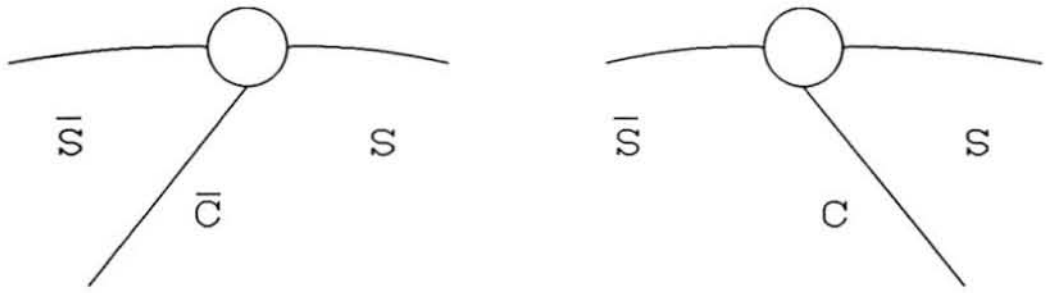


図3．4　ノードにおける方向
Fig.3.4 Directions of node.

［手法１］

（１）あるノードにおいて、リンクの障害のためデータが先に流せなくなった時、データの流す方向を次に示す方向に変更する。又、データの代替経路使用フラグ（初期値０）を１とし、代替経路カウンタを５とする。

ａ．偶数番ノードにおけるデータの初期駆動方向は次のとおり。

（ａ）s 方向へ進めない時 → 代替方向は $\bar{s}$ 方向

（ｂ）$\bar{s}$ 方向へ進めない時 → 代替方向は $\bar{c}$ 方向

（ｃ）$\bar{c}$ 方向へ進めない時 → 代替方向は s 方向

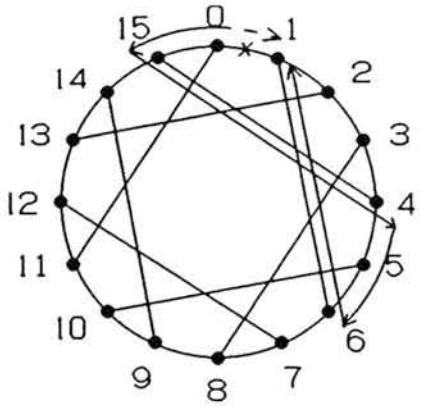
ｂ．奇数番ノードにおける、データの初期駆動方向は次のとおり。

（ａ）s 方向へ進めない時 → 代替方向は c 方向

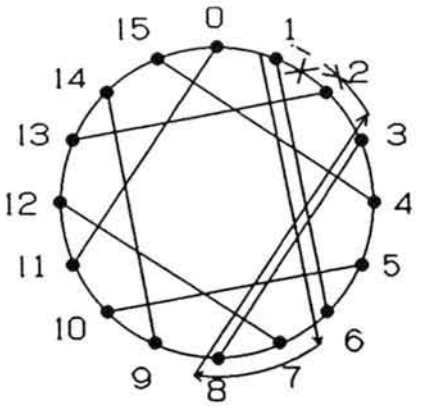
（ｂ）$\bar{s}$ 方向へ進めない時 → 代替方向は s 方向

（ｃ）c 方向へ進めない時 → 代替方向は $\bar{s}$ 方向

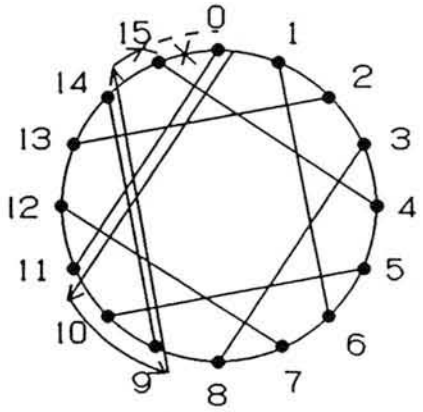
以上の６つの代替経路パターンについては、図３．５参照



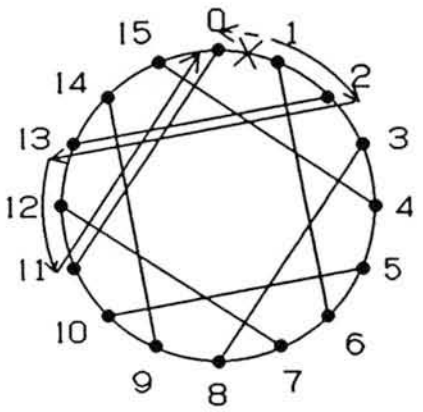
(a) S方向断



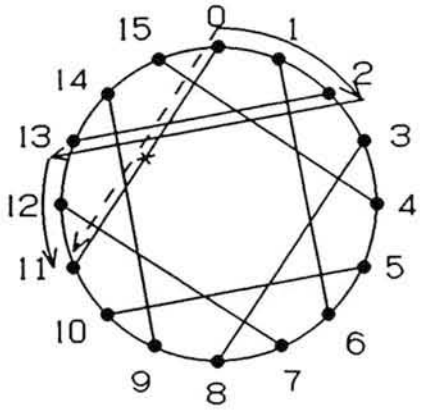
(a) S方向断



(b) $\bar{S}$方向断

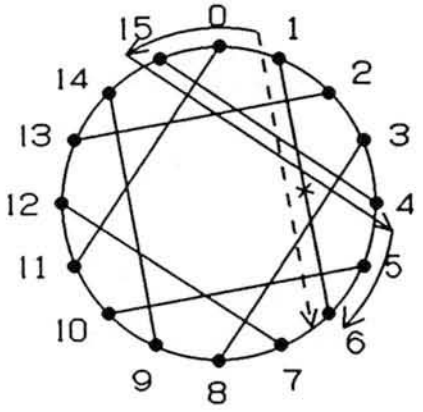


(b) $\bar{S}$方向断



(c) $\bar{c}$方向断

a 偶数ノードの先断



(c) c方向断

b 奇数ノードの先断

図3．5　6つの代替経路パターン

Fig.3.5 6 alternative route patterns.

（２）代替経路カウンタの値が０となるか、又はデータが目的地のノードに到着するまで次の処理を繰り返す。

代替経路使用フラグが１のデータを受け取ったノードは代替経路カウンタの値を１減ずると共に表３．２に示す代替連結表に従いデータを流す。

（３）代替経路カウンタの値が０となるか、又はデータが目的地に到着した場合、代替経路使用フラグを０とする。

例えば、図３．３においてノード０からノード６へデータが向かう場合、ノード１－６間が障害を起こした時、データはノード１において（１）ｂ．（ｃ）より c 方向へ進めないため $\bar{s}$ 方向のノード０へデータを送る。ノード０では表３．２に従い、s 方向のノード１から来たデータを $\bar{s}$ 方向のノード１５へ送る。ノード１５では表３．２に従い、s 方向のノード０から来たデータを c 方向のノード４へ送る。以下表３．２の代替連結表を用いて同様の処理を行い、データは１→０→１５→４→５→６と流れる。

手法１を多少改良したのが次に示す手法２がある。

［手法２］

（１）手法１の（１）において、代替経路開始時に、代替経路カウンタの値を３とする。

（２）手法１の（２）に同じ。

（３）手法１の（３）に同じ。

手法２においては、バタフライ形経路では代替経路長　$l=5$　であるため、３経路長だけ代替連結表で強制的にたどらせれば、後は通常の連結表でたどることができることを用いている。

手法２が手法１より優れている点は、代替経路の後半部で、最終的な目的地の方が代替経路終了点よりも近くなった場合に、目的地に直接行かせることができる点にある。

特殊な場合においては、距離を増加しないで済む代替経路が存在する。

例えば、図３．３において、ノード０からノード８へ行くには、

$$0 \rightarrow 15 \rightarrow 4 \rightarrow 3 \rightarrow 8$$

で距離４だが、次の様な代替経路でも同じ距離となる。

$$0 \rightarrow 1 \rightarrow 6 \rightarrow 7 \rightarrow 8$$

しかしながら、このように距離が同じ代替経路は一般には存在しない。

本章で示している手法は、あるリンクに対する最短の代替経路を求める手法であり、代替経路の局所的な最適化と言えよう。リンクの組み合わせからなる経路の最短性を保証するものではない。そのような経路を求めるには、データの持つ経路情報を増やすと共に、それに基づき各ノードで経路についての計算をし直す必要があるだろう。

3.2.4 障害リンクが2本ある場合

障害リンクが2本ある場合でも、代替経路上に2本目の障害リンクが無い場合には、**3.2.3**の手法をそのまま用いることができる。

しかしながら、代替経路上に2本目の障害リンクが存在した場合には、次に示す手法3を用いる必要がある。

手法3は、バタフライ形経路が各リンクに対して2本ずつ存在することを使用して、**3.2.3**で用いなかった方の代替経路を第2代替経路として用いるものである。なお、**3.2.3**で用いた代替経路を以後、第1代替経路と呼ぶ。データには、更に1 *bit* の第2代替経路使用フラグをつけ加える。

手法3では、まず第1代替経路上に障害リンクがあった場合、代替経路開始ノードにデータを戻す。これには、同じバタフライ経路上では、表3．3の第2代替連結表が、第1代替連結表と逆回りの関係にあることを使用する。

表3．3

表3．3 第2代替連結表

区分	入力方向	出力方向
偶数番ノード	$\bar{S}$	S
	S	$\bar{C}$
	$\bar{C}$	$\bar{S}$
奇数番ノード	$\bar{S}$	C
	S	$\bar{S}$
	C	S

次に、代替経路開始点より、第1代替経路で用いなかった方のバタフライ形経路の方向にデータを流し、以後は、再び第2代替連結表を用いて第2代替経路をたどらせる。

[手法3]

データが第1代替経路上を進む場合、あるノードより先に障害リンクがあり進めなくなった時には、次の手順を用いる。

(データの代替経路開始ノードへの戻し)

(1) データを直前の送信ノード方向のリンクに送り返すと共に、第2代替経路使用フラグを1とする。又、代替経路カウンタの値 k を $k=5-k$ とする。

(2) 代替経路カウンタの値が0となるまで次の処理を繰り返す。

第2代替経路使用フラグが1のデータを受け取ったノードは、代替経路カウンタの値を1減ずると共に表3．3の第2代替連結表に従いデータを流す。

(3) 代替経路カウンタの値が0になった所が、代替経路開始ノードである。第2代替経路使用フラグは1のままとする。

(第2代替経路のたどり)

(1) 代替経路カウンタの値を5とする。

a．偶数番目ノードにおけるデータの初期駆動方向は次のとおり。

(a) s 方向へ進めず、$\bar{s}$ 方向にも進めない時
→ 代替方向は $\bar{c}$ 方向

(b) $\bar{s}$ 方向へ進めず、$\bar{c}$ 方向にも進めない時
→ 代替方向は s 方向

(c) $\bar{c}$ 方向へ進めず、s 方向にも進めない時
→ 代替方向は $\bar{s}$ 方向

b．奇数番目ノードにおけるデータの初期駆動方向は次のとおり。

(a) s 方向へ進めず、c 方向にも進めない時
→ 代替方向は $\bar{s}$ 方向

(b) $\bar{s}$ 方向へ進めず、s 方向にも進めない時
→ 代替方向は c 方向

(c) c 方向へ進めず、$\bar{s}$ 方向にも進めない時
→ 代替方向は s 方向

(2) 代替経路カウンタの値が0となるか、又はデータが目的地のノードに到着するまで次の処理を繰り返す。

第2代替経路使用フラグが1のデータを受け取ったノードは代替経路カウンタの値を1減ずると共に、表3．3に示す第2代替連結表に従いデータを流す。

(3) 代替経路カウンタの値が0となるか、又はデータが目的地に到着した場合、第2代替経路使用フラグを0とする。

例えば、図３．６において、データを１から６へ流そうとする場合、リンク１―６及びリンク１―０が障害時、次のようにデータは流れる。

ノード１から６へ向かうデータは、リンク１―６（c 方向）に障害があるので、手法１に従い $\bar{s}$ 方向のノード０に向かおうとする。

ところが $\bar{s}$ 方向のリンク１―０に障害があり、この場合には、ノード１が代替経路開始ノードであるので、手法３の後半部に従い s 方向のノード２へ流す。以下、表３．３の第２代替連結表に従い　１→２→３→８→７→６と流れる。

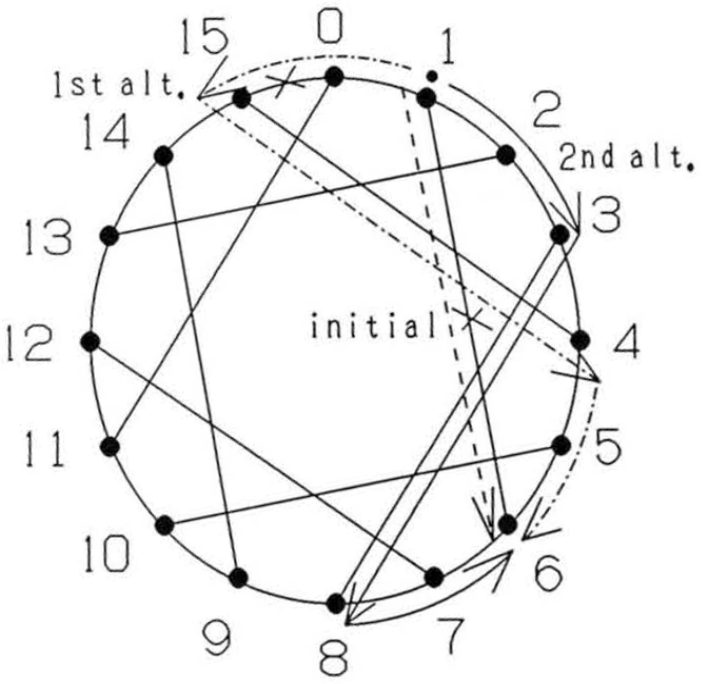


図3．6　第2代替経路

Fig.3.6 2nd alternative routes.

3.3 代替経路長の評価

バイトニック・ソートにおいて **3.2** の手法を用いた場合、実際の処理においてどのくらい経路長が増加するのか、求めてみることにする。

ネットワークのリンクの1ヶ所が障害を起こした場合、手法1を用いた時の経路長の合計は、障害のリンクの位置によって異なって来る。

なぜなら、バイトニック・ソートを行った時、ステップによって使うリンクと使わないリンクがあり、リンクによって使用頻度が異なるためである。

しかし、上限の目途は次のようにして求めることができる。

ある障害を起こしたリンクが全ステップに渡りいずれかの経路に影響を及ぼし、各ステップでは経路長が1から5へと4増加すると考えると、代替経路をたどった時の全経路長 L は次式により与えられる。

2.2.6 で示したようにノードのインデックスの差 2^i が $(w+1)/2$ より小さいうちは弧に沿った方が距離が短く $l_i = 2^i$ となり、$(w+1)/2$ を超える時には弦を用いた方が距離が短く $l_i = 2\lceil 2^i/(w+1)\rceil$ となることから、$2^k = (w+1)/2$ とすると、$k = (\log n)/2 - 1$ であり、$w+1 \approx \sqrt{n}$ より $2^{k+1} \approx \sqrt{n}$ となる。又、$n = 2^r$ より $r = \log n$ となる。

従って、

$$
\begin{aligned}
L &= \sum_{i=0}^{r-1} l_i \cdot f_i = \underbrace{\sum_{i=0}^{k} l_i \cdot f_i}_{\text{弧に沿う経路}} + \underbrace{\sum_{i=k+1}^{r-1} l_i \cdot f_i}_{\text{弦を使う経路}} \\
&= \sum_{i=0}^{k} (2^i + 4)(r - i) \\
&\quad + \sum_{i=k+1}^{r-1} (2\lceil 2^{i+1}/(w+1)\rceil + 4)(r - (i+1)) \\
&= \underbrace{\frac{1}{2}\sqrt{n}(\log n + 12) - 2\log n - 6}_{\text{障害のない時の経路長}} + \underbrace{2(\log n)^2 + 2\log n}_{\text{障害による経路長の増加}}
\end{aligned}
$$

l_i：各ステップでの経路長

f_i：経路長ごとの使用頻度

この値は、障害のない時の経路長に、$O((\log n)^2)$ が加わった形となっている。

つまり、n 個のノードからなる有限環結合ネットワークによるソーティングの全経路長は $O(\sqrt{n}\log n)$ であり、この値はリンクが1ヶ所障害を起こしても $O((\log n)^2)$ 程度増すだけである。

実際には、有弦環結合におけるバイトニック・ソートにおいては全ステップで用いるリンクは存在しないし、特に弦の障害時には、使用頻度から考え

経路長の増加は、より小さくなる。

4　ノード障害時のネットワーク型超並列コンピュータにおけるソーティング

バイトニック・ソートは超並列処理においてソーティングを行おうとする場合には効率的な手法である。しかし、ノードの1個でも障害を起こしていて使えない場合には、そのままのアルゴリズムでは一般的にはソーティングは不可能となる。

ここで、ノード障害時のソーティングとは、障害ノードを除いた状態でソーティングが行えることと定義する。つまり、障害ノードは使用不能とし、その番号は飛ばして考える。障害ノードの持つデータは使用不能とする。

このような場合には、一般的にはあらかじめ冗長なステップを設けておくと共に、障害ノードの番号により比較するデータの対の一部を変更することにより、ソーティングが実行可能となる。

しかしながら、特殊なケースにおいては、冗長なステップを加えたり、比較するデータの対を変更することなくソーティングを行うことが可能となる。

本章では、一般的なネットワーク型コンピュータにおいて、ノードの1個が障害を起こしてもバイトニック・ソートが行える手法と、その有弦環結合ネットワークへの適用方法を示す。

以下、**4.1** ではネットワーク型コンピュータのノード障害時のバイトニック・ソートの手法、**4.2** ではその改良法を示し、**4.3** では有弦環結合ネットワークへの適用方法を示す。

4.4 では特殊解であるノード障害時にステップ数を増やさずにバイトニック・ソートを行えるケースを示し、有弦環結合ネットワークの任意のノード障害に対し適用させる方法を示す。

4.1　ノード障害時のバイトニック・ソート

ネットワーク型コンピュータのノード障害時においてバイトニック・ソートを行うには、図4．1のバイトニック・ソートの処理状況を示すネットワークにステップを追加し処理に冗長性を持たせると共に、障害ノードの位置により比較するノードを一部変更することにより可能となる。この際、障害ノードは使用不能と考え、障害ノードの持つデータは処理の対象外とする。

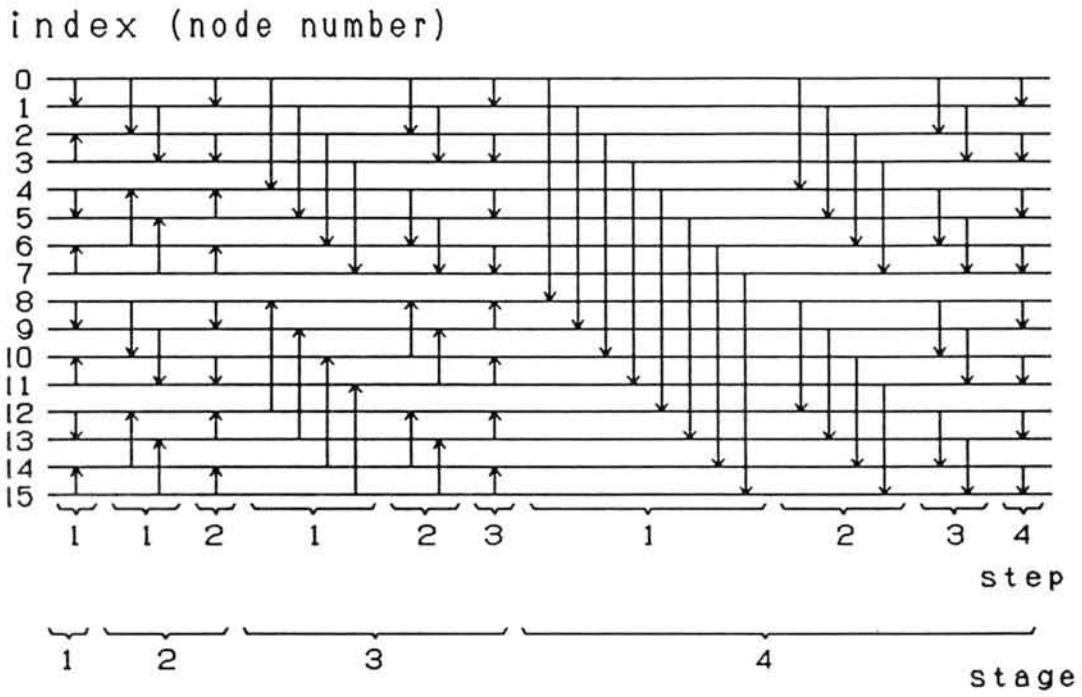


図4．1　バイトニック・ソートにおける処理状況　$n = 16$

Fig.4.1 A process of bitonic sort.

4.1.1 基本処理ネットワーク

ノード障害に対して耐故障性を持つバイトニック・ソート処理の基本形のネットワーク表現を図4．2に示す。

図4．1と比較すると、図4．2では冗長なステップが加わっている。

この基本形を基に、障害ノードの位置により **4.1.2** で述べるアルゴリズムにより各ステップで比較を行うノードのデータを変えることにより、つまり図4．2の縦線の長さを変えることにより、ノード障害時のバイトニック・ソートが可能になる。

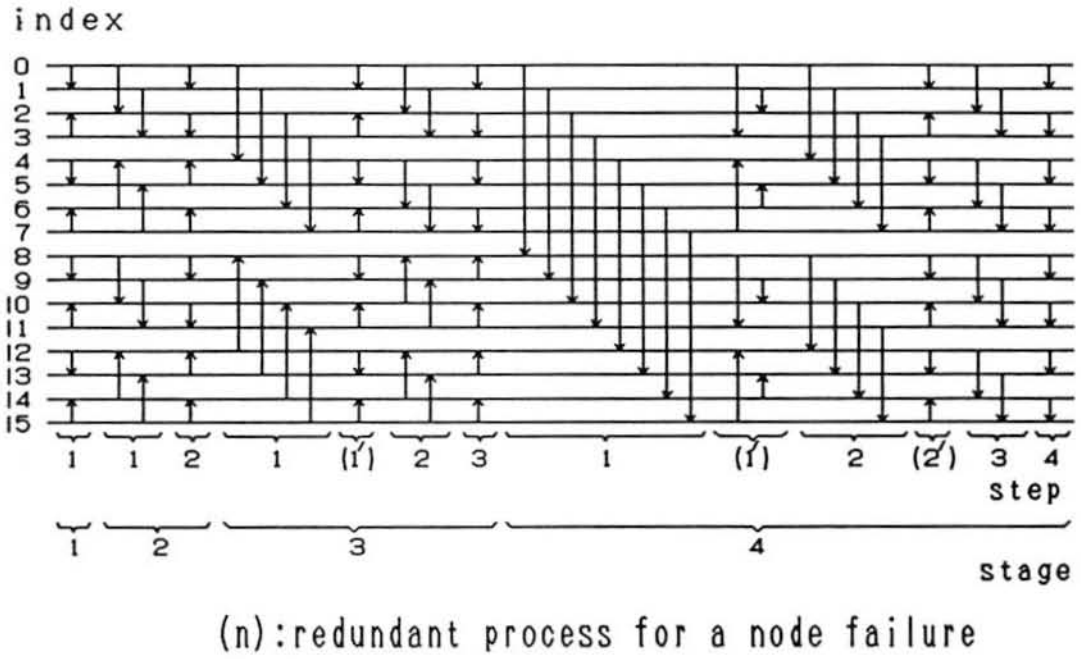


図4．2　ノード障害に対してフォールトトレラントな
バイトニック・ソートネットワークの基本形

Fig.4.2 A basic network for a fault-tolerant bitonic sort network with a node failure.

バイトニック・ソートのアルゴリズムは **2.2.5** で示したが、これをまとめるとつぎのように表すことができる。

[基本アルゴリズム]

ノード数を $n=2^r$ 個、ノードのインデックスを y とすると、i ステージ j ステップの処理は、次のようになる。但し、$i=1,2,\cdots,r,\quad j=1,2,\cdots,r$ であり $i \geq j$ とする。

（1）[送・受信ノードの指定] i ステージの j ステップにおいて、$\lfloor y/2^{i-j} \rfloor$ が偶数のノードは送信ノードで、奇数のノードは受信ノードである。但し、$0 \leq y \leq n-1$ であり、$\lfloor y \rfloor$ は y 以下の最大整数を表す。

（2）[データの移動] 送信ノードから、データをインデックス距離 $x=2^{i-j}$ だけ動かす。受信ノードでは、送信データの流れの妨げにならないように、データを退避させておく。

（3）[比較] 送信ノードからデータが到着すると、受信ノードでは、到着データと退避データとの比較が行われる。

比較後、受信ノードにおいて、$\lfloor y/2^i \rfloor$ が偶数の場合には値の小さい方のデータを送信ノードに戻し、奇数の場合には値の大きい方のデータを戻す。もう一方のデータは受信ノードで保持しておく。

図４．２は、このアルゴリズムに、次のようなアルゴリズムを追加することにより実現できる。

[追加アルゴリズム]

i ステージ j ステップにおいて、$j \leq i-2$ なるステップ j の後に次のステップを追加する。（これをステップ j' と呼ぶ）

（1）[送・受信ノードの指定] への追加

$y \ mod \ 2^{i-j-1} \leq 2^{i-j-2}-1$ の時、ノード y は送信ノード

そうでなければ、ノード y は受信ノード

（2）[データの移動] への追加

送信ノードから、データをインデックス距離

$x=2^{i-j-1}-2y \ mod \ 2^{i-j-1}-1$

だけ動かす。（図４．３参照）

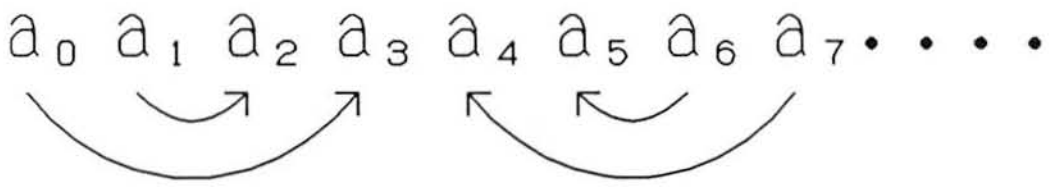


図4．3　$i = 4, j = 1$ 後の追加ステップ　$(n = 16)$
Fig.4.3 Additional step after i=4,j=1　$(n = 16)$

（3）[比較] への追加

受信ノードにおいて比較後、次のデータを送信ノードに戻す。

$y \ mod \ 2^{i-j} \leq 2^{i-j-1} - 1$ の時、

小さい方のデータ

そうでなければ、大きい方のデータ

ステップ数については、次のようにして求めることができる。

図4．1に示したオリジナルのバイトニック・ソートにおいては、データ数 $n = 2^r$ 個の時、ステージ i におけるステップ数は i であり、ステップ数の合計 S は、

$$S = \sum_{i=1}^{r} i = \frac{1}{2} r(r+1) = \frac{1}{2} \log n(\log n + 1)$$

となる。

一方、図4．2においては、ステージ i $(i \geq 2)$ におけるステップ数は、$i + (i-2) = 2i - 2$ であり、ステップ数の合計 S は、

$$S = 1 + \sum_{i=2}^{r} (2i - 2) = (\log n)^2 - \log n + 1$$

となり、図4．1の約2倍のステップ数となる。

4.1.2 移動距離変更アルゴリズム

インデックスが k 番目のノードが障害を起こした場合には、送信ノードから動くデータのノード間のインデックス距離 x は、インデックス y のノードにおいて **4.1.1** [基本アルゴリズム] の（2）で示した値から次のように変わる。

（a）$k \ mod \ 4x = 0$ のとき、

全てのノードにおいて x の値はそのまま。

（b）$0 < k \ mod \ 4x < x$ のとき、

$\lfloor \frac{k}{4x} \rfloor * 4x \leq y < k$ なる y において、

$x \rightarrow x + 1$

とする。

（c）$k \ mod \ 4x = x$ のとき、

全てのノードにおいて x の値はそのまま。

（d）$x < k \ mod \ 4x < 2x$ のとき、

（i）$y = \lfloor \frac{k}{4x} \rfloor * 4x$ なる y において、

$x \rightarrow 0$

（ii）$\lfloor \frac{k}{4x} \rfloor * 4x < y \leq k - x$ なる y において、

$x \rightarrow x-1$

とする。

（e）$2x \leq k \ mod \ 4x < 3x-1$ のとき、

$k < y \leq \lceil \frac{k}{4x} \rceil * 4x - (x+1)$ なる y において、

$x \rightarrow x-1$

とする。

（f）$k \ mod \ 4x = 3x-1$ のとき、

全てのノードにおいて x の値はそのまま。

（g）$3x \leq k \ mod \ 4x < 4x-1$ のとき、

（i）$k-x \leq y < k-1$ なる y において、

$x \rightarrow x+1$

（ii）$y = \lceil \frac{k}{4x} \rceil * 4x - (x+1)$ なる y において、

$x \rightarrow 0$

とする。

（h）$k \ mod \ 4x = 4x-1$ のとき、

全てのノードにおいて x の値はそのまま。

特に、$x=1$ の場合には、上記アルゴリズムからわかるようにいかなるケースでも x の値はそのままで良い。

又、$k=0$ の場合には、x の値は全てのノード y において変わらない。

なお、$y=k$ のノードに対しては、データの送受信は行わないものとする。

例えば、ノード数 $n=16$ で障害ノード $k=3$ の場合には、ノード y におけるデータの移動するインデックス距離 x は次のようになる。（図4．4（a）参照）

$x=2$ のとき、（d）より、

$y=0$ で　　$x=2 \rightarrow 0$

$y=1$ で　　$x=2 \rightarrow 1$

$x=4$ のとき、（b）より、

$y=0,1,2$ で　　$x=4 \rightarrow 5$

$x=8$ のとき、（b）より、

$y=0,1,2$ で　　$x=8 \rightarrow 9$

又、$n=16$,　$k=3$ の場合は次のようになる。（図4．4（b）参照）

$x=2$ のとき、（h）より、

全ての y において x の値はそのまま。

$x=4$ のとき、（d）より、

$y=0$ で　　$x=4 \rightarrow 0$

$y=1,2,3$ で　　$x=4 \rightarrow 3$

$x=8$ のとき、（b）より、

$y=0,1,2,3,4,5,6$ で　　$x=8 \rightarrow 9$

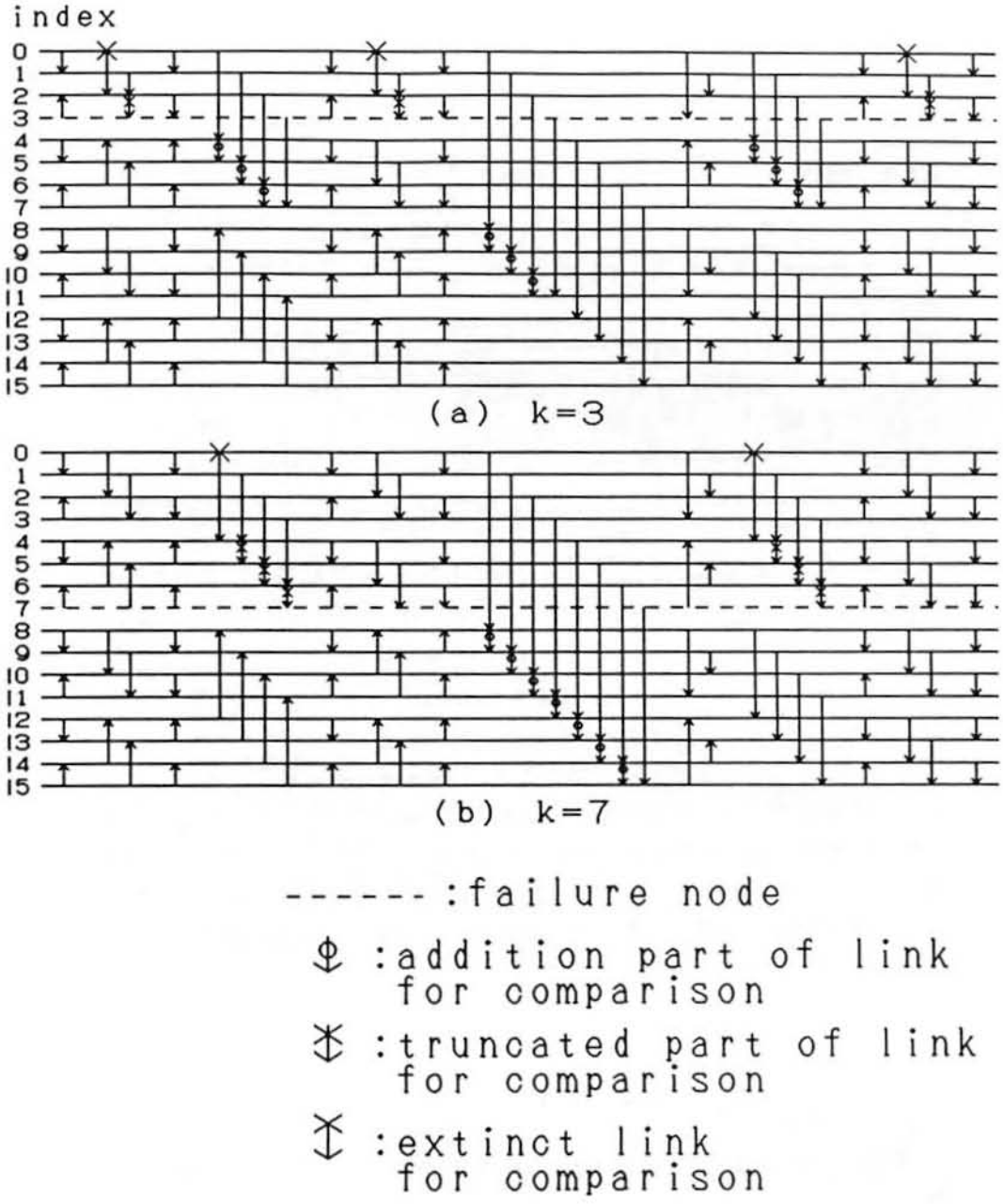


図4.4 ノード障害時のバイトニック・ソートの例

Fig.4.4 Examples of bitonic sort with a node failure.

4.1.3 証明

バイトニック・ソートにおいては、次の手順を繰り返している。
（1）2^i 個のデータの単調列を2組ずつ組み合わせて、2^{i+1} 個のデータの双調列を作る。
（2）2^{i+1} 個のデータの双調列から 2^{i+1} 個のデータの単調列を作る。この際、**2.2.1** で述べた双調列の性質をうまく使用して、2^{i+1} 個のデータの双調列から2組の 2^i 個のデータの双調列、さらにそこから 2^2 組の 2^{i-1} 個のデータの双調列、… と再帰的に繰り返し、2^{i+1} 個のデータの単調列を生成する。

（1）、（2）を繰り返すことにより、最終的に n 個のデータの単調列を生成する。

ところで、2^{i+1} 個のデータの双調列から2組の 2^i 個のデータの双調列を生成すると、**2.2.1** で述べたバイトニック・ソートの性質により、大きい方の 2^i 個のデータの双調列と小さい方の 2^i 個のデータの双調列とに分かれる。ノード障害が無い場合には、そのまま再帰的に、2^{i-2} 個 2^{i-3} 個、… と短い双調列を作って行けば最後に単調列が完成する。

ところが、ノード障害が有る場合には、再帰性を単純に使うことができない。2^{i+1} 個のデータの双調列から、2組の 2^i 個のデータに分ける際、一方の組の各値が他方の組の各値を超えることがないようにするために、**4.1.1**のアルゴリズムを追加すると共に、**4.1.2** に示したように比較するデータのインデックス距離 x の値を障害ノードの位置により変える必要がある。

以下にその証明を示す。

双調列 $(a_0, a_1, a_2, \cdots, a_{x-1}, a_x, \cdots, a_{2x-1})$ を、

（a）$(a_0, a_1, a_2, \cdots, a_{x-1})$

（b）$(a_x, a_{x+1}, \cdots, a_{2x-1})$

と2分割して考える。

障害ノード k は（a）又は（b）のいずれかにあり、データ a_k は使用不能とする。

（1）$0 < k \bmod 4x < x$ の場合

障害ノード k は、（a）側にある。

a．双調列の最大値が（b）側の時

双調列に次のような関係がある時、

$a_0 \le a_1 \le a_2 \le \cdots \le a_k \le \cdots \le a_{x-1}$ …（a）

$\le a_x \le \cdots \le a_i \ge a_{i+1} \ge \cdots \ge a_{2x-1}$ …（b）

a_x と a_{2x-1} とのうち小さい方の値は、（b）側の最小値となっている。

$\min(a_x, \cdots a_i, a_{i+1}, \cdots, a_{2x-1}) = a_x \ or \ a_{2x-1}$

a_x と a_{2x-1} とのうち大きい方の値は、昇べき順に数えて（a_k を除いて）x 番目以上の値となっている。

なぜなら、

$a_0 \leq a_1 \leq \cdots \leq a_{x-1} \leq a_x \leq \max(a_x, a_{2x-1})$

であるから。

そこで、a_x と a_{2x-1} とをまず比較し、$\max(a_x, a_{2x-1})$ をノード x の位置に、$\min(a_x, a_{2x-1})$ をノード $2x-1$ の位置に持って来る。この時双調列の関係をくずさないようにするために追加ステップのような処理を行う。この処理は次の2分割処理の準備となっている。

ノード x の値 $\max(a_x, a_{2x-1})$ は、昇べき順に数えて x 番目以上の値であるから、この双調列を2つに分けた時、(b)側に残っていなければならない。

そこで、ノード x の値 $\max(a_x, a_{2x-1})$ を除いた(b)側の $x-1$ 個の値と障害ノード k の値 a_k を除いた(a)側の $x-1$ 個の値とを各々2個ずつペアにして比較を行い、各々小さい方の値を(a)側に、大きい方の値を(b)側に持って来るようにする。

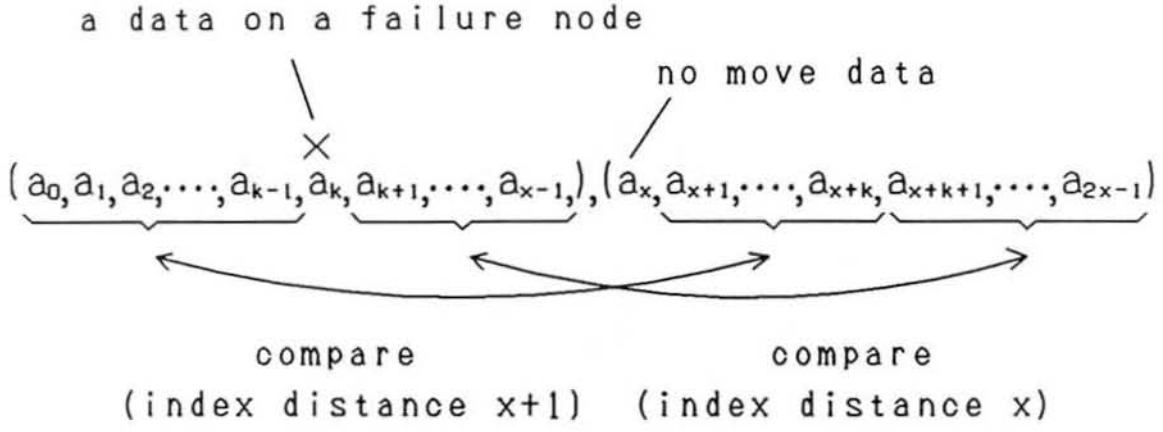
そのためには、(a)側の $(a_0, a_1, \cdots a_{k-1})$ は、(b)側の $(a_{x+1}, a_{x+2}, \cdots a_{x+k})$ と比較するようにし、移動するインデックス距離を $x \rightarrow x+1$ とする。

(a)側の $(a_{k+1}, a_{k+2}, \cdots a_{x-1})$ については、障害ノードが無い場合と同様(b)側の $(a_{x+k+1}, \cdots a_{2x-1})$ と比較すれば良く、移動するインデックス距離は x のままで良い。

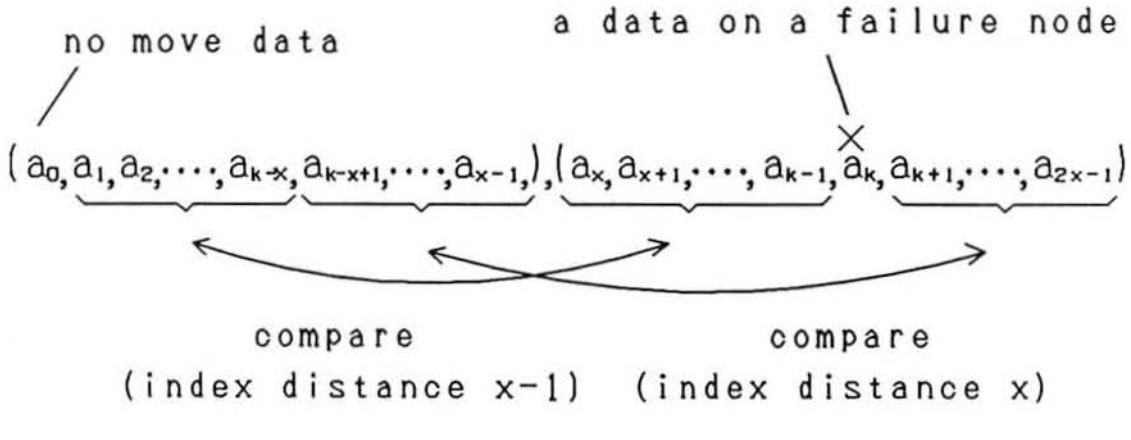
つまり、 $\lfloor \frac{k}{4x} \rfloor * 4x \leq y < k$ なる y において、

$x \rightarrow x+1$

とすれば良い。(図4．5(a)参照)



(a) $0<k \bmod 4x<x$



(b) $x<k \bmod 4x<2x$

図4．5　ノード障害時のデータ比較
Fig.4.5 Date comparison with a failure node.

b．双調列の最大値が（a）側の時

双調列に次のような関係がある時、

$$a_0 \leq a_1 \leq \cdots \leq a_k \leq \cdots \leq a_i \geq a_{i+1} \geq \cdots \geq a_{x-1} \quad \cdots(a)$$

$$\geq a_x \geq a_{x+1} \geq \cdots\cdots\cdots \geq a_{2x-1} \quad \cdots(b)$$

$\max(a_x, a_{2x-1}) = a_x$ は、（b）の関係、

$a_{2x-1} \leq \cdots\cdots \leq a_{x+1} \leq a_x$

より、昇べき順に数えて、x 番目以上の値であるので、（a）側と（b）側との値の比較後も（b）側に残らなければならない。

よって、a．のケースと同じアルゴリズムを用いることができる。

（2）$x < k \bmod 4x < 2x$ の場合

障害ノード k は、（b）側にある。

a．双調列の最大値が（a）側にある時

双調列に次のような関係がある時、

$$a_0 \leq a_1 \leq a_2 \leq \cdots \leq a_i \geq a_{i+1} \geq \cdots \geq a_{x-1} \quad \cdots(a)$$

$$\geq a_x \geq \cdots \geq a_k \geq \cdots\cdots \geq a_{2x-1} \quad \cdots(b)$$

a_0 と a_{x-1} とのうち小さい方の値は、（a）側の最小値となっている。

$\min(a_0, a_1, \cdots a_i, a_{i+1}, \cdots, a_{x-1}) = a_0 \ or \ a_{x-1}$

従って、$\min(a_0, a_{x-1})$ 以上の値が $x-1$ 個（a）側に存在する。

そこで、a_0 と a_{x-1} とをまず比較し、$\min(a_0, a_{x-1})$ をノード 0 の位置に、$\max(a_0, a_{x-1})$ をノード $x-1$ の位置に持って来る。この時双調列の関係がくずれないようにするために追加ステップのような処理を行う。この処理は次の2分割処理の準備となっている。

ノード 0 の値 $\min(a_0, a_{x-1})$ は、それ以上の値が少なくとも $x-1$ 個存在するのでこの双調列を2つに分けた時、（a）側に残っていなければならない。

そこで、ノード 0 の値 $\min(a_0, a_{x-1})$ を除いた（a）側の $x-1$ 個の値と障害ノード k の値 a_k を除いた（b）側の $x-1$ 個の値とを各々2個ずつペアにして比較を行い、各々小さい方の値を（a）側に、大きい方の値を（b）側に持って来るようにする。

そのためには、（a）側の（$a_1, a_2, \cdots a_{k-x}$）は、（b）側の（$a_x, a_{x+1}, \cdots a_{k-1}$）と比較するようにし、移動するインデックス距離を $x \rightarrow x-1$ とする。

又、（a）側の a_0 については $x \rightarrow 0$ とし動かさない。

なお、（a）側の（$a_{k-x+1}, a_{k-x+2}, \cdots a_{x-1}$）については移動するインデックス距離は x のままで良い。（図4．5（b）参照）

b．双調列の最大値が（b）側にある時

双調列に次のような関係がある時、

$a_0 \leq a_1 \leq a_2 \leq \cdots\cdots\cdots\cdots \leq a_{x-1}$ ・・・（ａ）

$\leq a_x \leq a_{x+1} \leq \cdots \leq a_k \leq \cdots \leq a_i \geq \cdots \geq a_{2x-1}$ ・・・（ｂ）

$\min(a_0, a_{x-1})$ は、（ａ）の関係、

$a_0 \leq a_1 \leq a_2 \leq \cdots\cdots \leq a_{x-1}$

より、それ以上の値が少なくとも $x-1$ 個存在するので、（ａ）側と（ｂ）側との値の比較後も（ａ）側に残らなければならない。

よって、ａ．のケースと同じアルゴリズムを用いることができる。

（３）$k\ mod\ 4x = 0, x$ の時

$k\ mod\ 4x = 0$ では、$y = k$ のデータは（ａ）側で障害ノードにあるために、ノード $y = k + x$ に流すことができない。

そこで、ノード y からの移動インデックス距離 x をそのままにしておいても、比較後 a_x は（ｂ）側に残すことができる。

$k\ mod\ 4x = x$ では、$y = k$ のデータは（ｂ）側で障害ノードにあたるために、ノード y からの移動インデックス距離 x をそのままにしておいても、比較後 a_0 は（ａ）側に残すことができる。

他の場合も同様にして証明できる。

4.2 ノード障害時のバイトニック・ソート（改良法）

この節では、**4.1** の手法の改良法について述べる。

この手法（以下、新手法）は、**4.1** の手法（以下、前手法）における基本処理ネットワークを簡略化したものである。

4.2.1 基本処理ネットワーク（改良型）

ノード障害に対して耐故障性を持つバイトニック・ソート処理の基本形（改良型）のネットワーク表現を図４．６に示す。

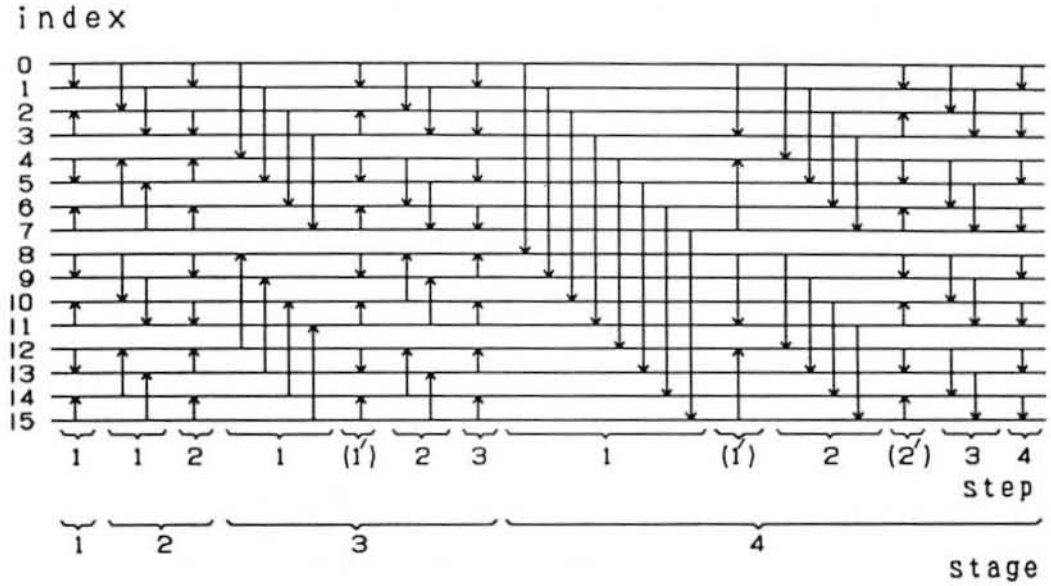
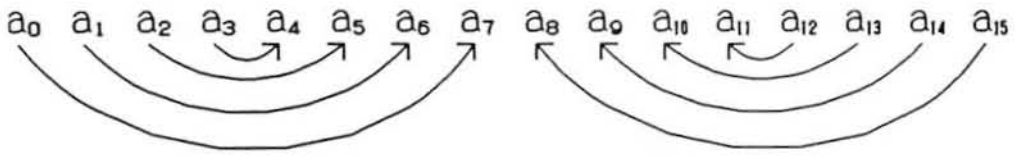


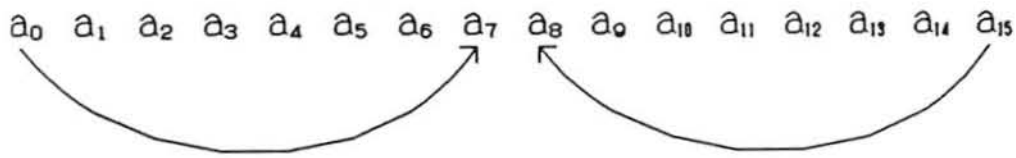
図4.6　ノード障害に対してフォールトトレラントな
バイトニック・ソートネットワークの基本形（改良型）

Fig.4.6 A basic network for a fault-tolerant bitonic sort network with a node failure (revised version).

図４．１と比較すると、図４．６では追加ステップが入っている。この追加ステップは、前手法では図４．７（ａ）のようなノード間の比較を行なっていたが、新手法では図４．７（ｂ）のようなノード間の比較のみとしている。但し、図では、前手法と新手法の違いを明らかにするために、データ数 $n = 32$ での $i = 5, j = 1$ 後の追加ステップでの比較処理を示している。



(a) old method



(b) new method

図4．7　$i=5, j=1$ 後の追加ステップ　$(n=32)$

Fig.4.7 Additional step after i=5,j=1　$(n=32)$

この基本形を基に、障害ノードの位置により **4.1.2** で示したのと同じ移動距離変更アルゴリズムにより各ステップで比較を行うノードの対を変えることにより、つまり図４．６の縦線の長さを変えることにより、ノード障害時のバイトニック・ソートが可能になる。

図４．６は、**4.1.1** で示したバイトニック・ソートの基本アルゴリズムに、次のようなアルゴリズムを追加することにより実現できる。

[追加アルゴリズム（改良法）]

i ステージ j ステップにおいて、$j \leq i-2$ なるステップ j の後に１ステップずつ次のアルゴリズムを追加する。(これをステップ j' と呼ぶ)

（１）[送・受信ノードの指定] への追加

$y \ mod \ 2^{i-j-1} = 0$ の時、ノード y は送信ノード

$y \ mod \ 2^{i-j-1} = 2^{i-j-1} - 1$ の時、ノード y は受信ノード

（２）[データの移動] への追加

送信ノードから、データをインデックス距離

$x = 2^{i-j-1} - 1$

だけ動かす。(図４．７（ｂ）参照)

（３）[比較] への追加

受信ノードにおいて比較後、次のデータを送信ノードに戻す。

$y \ mod \ 2^{i-j} = 2^{i-j-1} - 1$ の時、

小さい方のデータ

そうでなければ、大きい方のデータ

並列的にみたステップ数の合計 S は、ステージ $i(i \geq 2)$ におけるステップ数が、$i + (i-2) = 2i - 2$ であることから、

$$S = 1 + \sum_{i=2}^{r} (2i - 2) = (\log n)^2 - \log n + 1$$

となり、図４．１の約２倍のステップ数となる。

これは論理的に並列に考えると前手法と同じステップ数になる。

しかし、ネットワークの形状が完全結合でない限りは、[7]通信密度から考えて新手法の方が優れている。

新手法によれば、例えば、ノード数 $n = 16$ で障害ノード $k = 3$ の場合には、ノード y におけるデータの移動するインデックス距離 x は次のようになる。(図４．８参照)

$x = 2$ のとき、(ｄ) より、

$y = 0$ で　　$x = 2 \rightarrow 0$

$y = 1$ で　　$x = 2 \rightarrow 1$

[7]結合方式については、文献（３）、（４）参照。

$x=4$ のとき、(b) より、

$y=0,1,2$ で　　$x=4 \to 5$

$x=8$ のとき、(b) より、

$y=0,1,2$ で　　$x=8 \to 9$

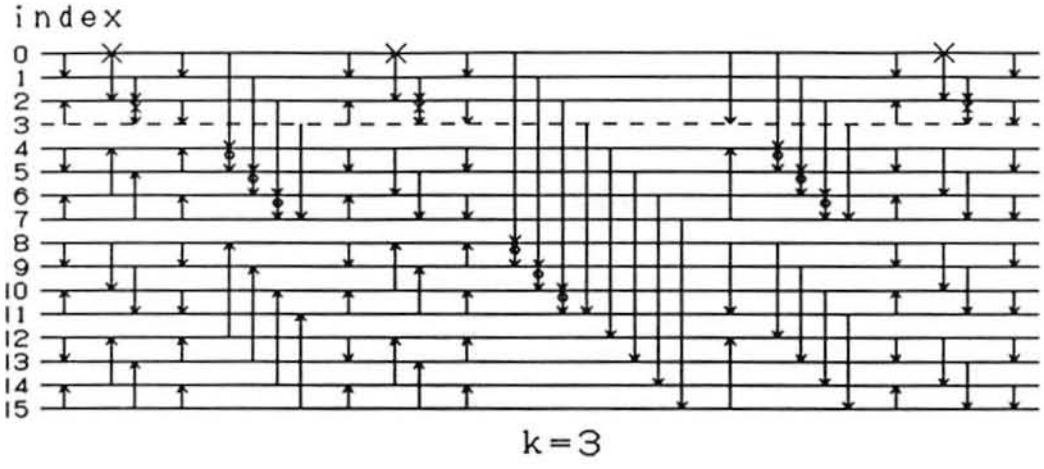


図4．8　ノード障害時のバイトニックソートの例

Fig.4.8 An example of bitonic sort with a node failure.

4.2.2 証明

以下に新手法の証明を示す。

双調列 $(a_0, a_1, a_2, \cdots, a_{x-1}, a_x, \cdots, a_{2x-1})$ を、

（ａ）$(a_0, a_1, a_2, \cdots, a_{x-1})$

（ｂ）$(a_x, a_{x+1}, \cdots, a_{2x-1})$

と２分割して考える。

障害ノード k は（ａ）又は（ｂ）のいずれかにあり、データ a_k は使用不能とする。

（１）$0 < k \ mod \ 4x < x$ の場合

障害ノード k は、（ａ）側にある。

ａ．双調列の最大値が（ｂ）側の時

まず、追加ステップの証明を行う。

双調列に次のような関係がある時、

$a_0 \leq a_1 \leq a_2 \leq \cdots \leq a_k \leq \cdots \leq a_{x-1}$ ・・・（ａ）

$\leq a_x \leq \cdots \leq a_i \geq a_{i+1} \geq \cdots \geq a_{2x-1}$ ・・・（ｂ）

a_x と a_{2x-1} とのうち小さい方の値は、（ｂ）側の最小値となっている。

$\min(a_x, \cdots a_i, a_{i+1}, \cdots, a_{2x-1}) = a_x \ or \ a_{2x-1}$

a_x と a_{2x-1} とのうち大きい方の値は、昇べき順に数えて（a_k を除いて）x 番目以上の値となっている。

なぜなら、

$a_0 \leq a_1 \leq \cdots \leq a_{x-1} \leq a_x \leq \max(a_x, a_{2x-1})$

であるから。

そこで、a_x と a_{2x-1} とをまず比較し、$\max(a_x, a_{2x-1})$ をノード x の位置に、$\min(a_x, a_{2x-1})$ をノード $2x-1$ の位置に持って来る。

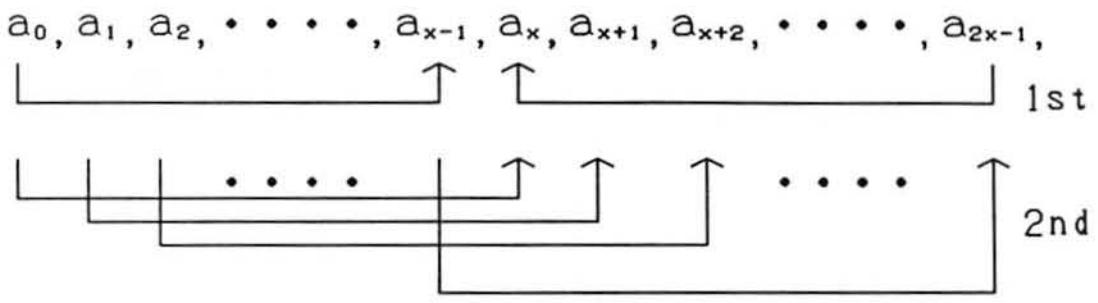
この時、前手法では、a_x, a_{2x-1} 以外のデータについても双調列の関係をくずさないようにするために、図４．７（ａ）のような比較を行っていた。しかし、図４．７（ｂ）のように a_x, a_{2x-1} 以外のデータの比較は不要であることが判明した。（図４．７では、前手法、新手法とも a_x, a_{2x-1} 側のデータの比較を行うと共に、a_0, a_{x-1} 側のデータの比較も行っているが、これは、障害ノード k が（ａ）側でも（ｂ）側でも成り立つようにこのステップに汎用性を持たせるためである。）

本来のバイトニック・ソート（障害ノードがない時のバイトニック・ソート）においては、**2.2.1** に示したように、双調列 $(a_0, a_1, \cdots a_{2x-1})$ において、次のような２つの順序列（双調列）を再帰的に作って行く。

$$(\min(a_0, a_x), \min(a_1, a_{x+1}), \cdots\cdots, \min(a_{x-1}, a_{2x-1}))$$

$$(\max(a_0, a_x), \max(a_1, a_{x+1}), \cdots\cdots, \max(a_{x-1}, a_{2x-1}))$$

一方、本章に示す障害ノードがある時のバイトニック・ソートにおいては、上記処理の前に追加ステップの処理を行っている。(図4．9参照)



1st: Additional step

2nd: Original step

図4．9　障害ノードがある場合のバイトニックソート

Fig.4.9 Bitonic sort with a node failure.

その結果、各ノードには、次のようなデータが入る。

ノード 0：$\min(a_0, a_{x-1})$, ノード $x-1$：$\max(a_0, a_{x-1})$

ノード x：$\max(a_x, a_{2x-1})$, ノード $2x-1$：$\min(a_x, a_{2x-1})$

次に本来のバイトニック・ソートの処理である双調列の分割に入る。

ここでは、前処理を行った後に双調列の分割を行っても、順序列は、min 側と max 側とに分かれることを証明する。

まず、前処理においては、a_0, a_{x-1} については、（a）より $a_0 \leq a_{x-1}$ であるから位置関係は変わらない。a_x, a_{2x-1} については、

（i） $a_x \geq a_{2x-1}$ の時、

a_x, a_{2x-1} の位置関係は変わらない。

従って、次の双調列の分割のステップでは、本来どおりの処理が行われ、min 側と max 側に分かれる。

（ii） $a_x < a_{2x-1}$ の時、

前処理の後、

ノード x： a_{2x-1}

ノード $2x-1$： a_x

となり、次の双調列の分割のステップでは、a_0 と a_{2x-1}、a_{x-1} と a_x の比較が行われることになる。（他のデータについては、本来のバイトニック・ソートどおりの比較が行われる。）

a_{x-1} と a_x との比較については、（a），（b）より $a_{x-1} \leq a_x$ であるから、位置関係はそのままで、min 側に a_{x-1} 、max 側に a_x が来る。

つまり、ノード $0, x-1, x, 2x-1$ のデータに注目すると、

$$\left.\begin{array}{l}\text{min 側：ノード } 0(\min(a_0, a_{2x-1})),\ \text{ノード } x-1(a_{x-1}) \\ \text{max 側：ノード } x(\max(a_0, a_{2x-1})),\ \text{ノード } 2x-1(a_x)\end{array}\right\}(c)$$

となる。

一方、オリジナルのバイトニック・ソートでは、

$$\left.\begin{array}{l}\text{min 側：ノード } 0(a_0),\ \text{ノード } x-1(\min(a_{x-1}, a_{2x-1})) \\ \text{max 側：ノード } x(a_x),\ \text{ノード } 2x-1(\max(a_{x-1}, a_{2x-1}))\end{array}\right\}(d)$$

となる。

（c）と（d）とを比較すると、

$$a_0 \leq a_{x-1} \leq a_x < a_{2x-1}$$

であるから、

（c）は、

$$\left.\begin{array}{lll}\text{min 側：} & \text{ノード } 0(a_0), & \text{ノード } x-1(a_{x-1}) \\ \text{max 側：} & \text{ノード } x(a_{2x-1}), & \text{ノード } 2x-1(a_x)\end{array}\right\}(c')$$

となる。

一方、（d）は、

min 側： ノード $0(a_0)$, ノード $x-1(a_{x-1})$
max 側： ノード $x(a_x)$, ノード $2x-1(a_{2x-1})$ $\Big\}(d')$

となる。

（c）′、（d）′より、追加ステップが加わっても、次の双調列の分割のステップで正しく min 側、max 側に分かれることが証明されたことになる。

次に、双調列を min 側、max 側とに分割する部分のアルゴリズムの証明が必要となるが、この部分の証明は、**4.1.3** と同じなので省略する。

b．双調列の最大値が（a）側の時

双調列に次のような関係がある時、

$$a_0 \leq a_1 \leq \cdots \leq a_k \leq \cdots \leq a_i \geq a_{i+1} \geq \cdots \geq a_{x-1} \quad \cdots(a)$$

$$\geq a_x \geq a_{x+1} \geq \cdots\cdots\cdots \geq a_{2x-1} \quad \cdots(b)$$

$\max(a_x, a_{2x-1}) = a_x$ は、（b）の関係、

$$a_{2x-1} \leq \cdots\cdots \leq a_{x+1} \leq a_x$$

より、昇べき順に数えて、x 番目以上の値であるので、（a）側と（b）側との値の比較後も（b）側に残らなければならない。

よって、a．のケースと同じアルゴリズムを用いることができる。

$x < k \bmod 4x < 2x$ の場合等、他の場合も同様にして証明できるので省略する。

4.3 有弦環結合ネットワークへの適用

4.1、**4.2**では論理的なネットワーク上でのノード障害時のバイトニック・ソートの手法について述べた。ここでは、この手法を物理的なネットワーク上に適用してみる。ネットワークの例として有弦環結合ネットワークを取り上げる。

4.3.1 障害ノードの検出

4.1、**4.2**の手法では、各ステップにおいてデータを流す際に、目的地のノード番号をデータに与えてやる必要がある。このためには、ソーティングを行う前に事前処理で障害ノードを検出して各ノードに障害ノード番号を通知する必要がある。この処理は、例えば隣接するノードをウオッチドッグ・タイマで監視しておき障害を検出すると、その情報（障害ノード番号）をブロードキャストするという方法がある。

4.3.2 障害ノードの迂回

障害ノードがある場合、完全結合でない限りは障害ノードがデータの中継ノードとなっていることがある。このような場合、データは障害ノードを迂

回するような代替経路をとらなければならない。

有弦環結合ネットワークにおいては、代替経路を図４．１０に示すような８の字型に規則的に指定することができる。例えば、データをノード０→１→６と流そうとした場合において、ノード１が障害を起こしていて、ノード０から１へデータを流すことができないときには、代替経路を用いてノード０→１５→４→５→６とデータを流すことができる。この代替経路長 l_d は、ノード障害においては弦の長さ w によらず常に一定（$l_d = 4$）となる。[8]

代替経路は、各ノードに代替経路表を持たせておくことにより実現できる。ノード障害時には、障害ノードに隣接するノードにおいて代替経路表を用いることにより障害ノードを迂回させる。

[8]一方、リンク障害時には **3** で示したように $l_d = 5$ となる。

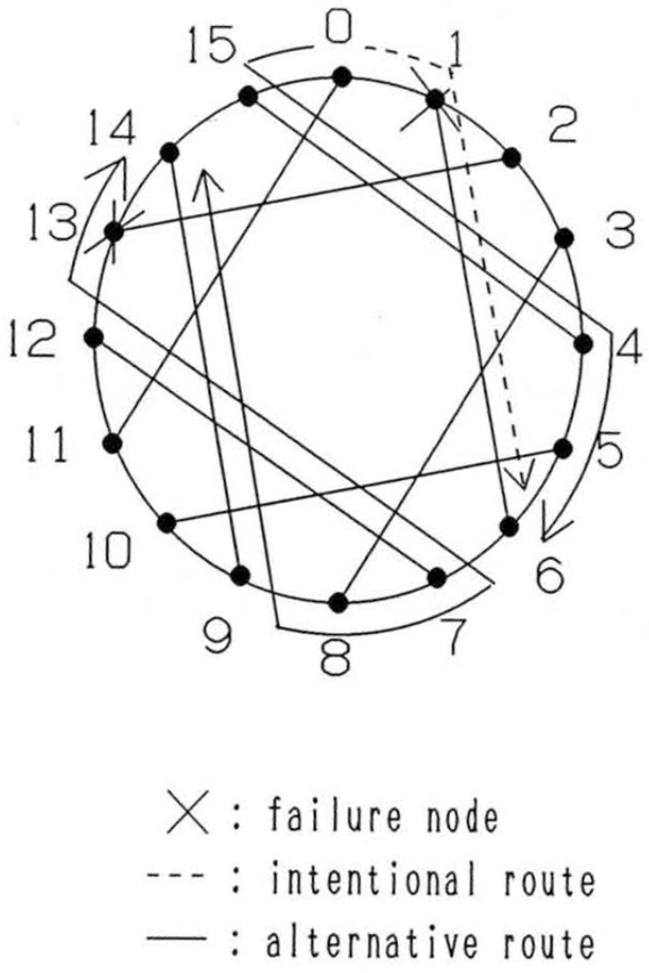


図4．10　ノード障害時の代替経路
Fig.4.10 Alternative routes of node failure.

4.4 ステップ数を増やさずにできる方法

障害ノードのあるネットワーク型コンピュータ上でバイトニック・ソートを行おうとする場合、一般的に任意の障害ノードに対しては冗長ステップの設定と、障害ノードの位置により比較するデータの一部の対を変更するためインデックス距離 x の変更が必要となる。

ところが、図4．11のタイプのバイトニック・ソート（図4．1と同じ）においては、障害ノードが先頭の0番目の場合には、冗長ステップを設けたり、インデックス距離 x を変更することなしにソーティングを行うことが可能となる。

又、バイトニック・ソートの別のタイプのアルゴリズムである図4．12の場合には、最後の $n-1$ 番目のノードが障害ノードの時には、そのままソーティングを行うことができる。

図4．12は、ノード数 $n=2^r$ の場合、$r-1$ ステージ以下の時には、各受信ノードにおいて2つのデータを比較した後各送信ノードに戻すデータの大小関係を図4．11のタイプと逆にしたものである。

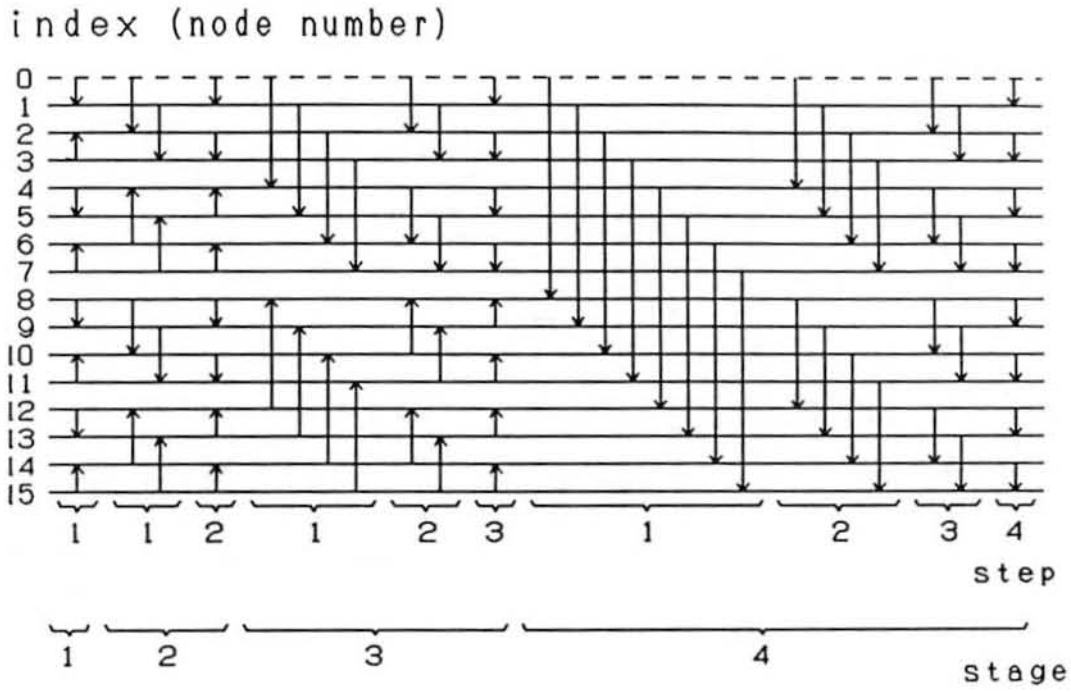


図4．11　バイトニック・ソートにおける処理状況　$n = 16$
（0番目のノード障害に適用できるバイトニック・ソート）
Fig.4.11 A process of bitonic sort. (Bitonic sort for the 0th node failure.)

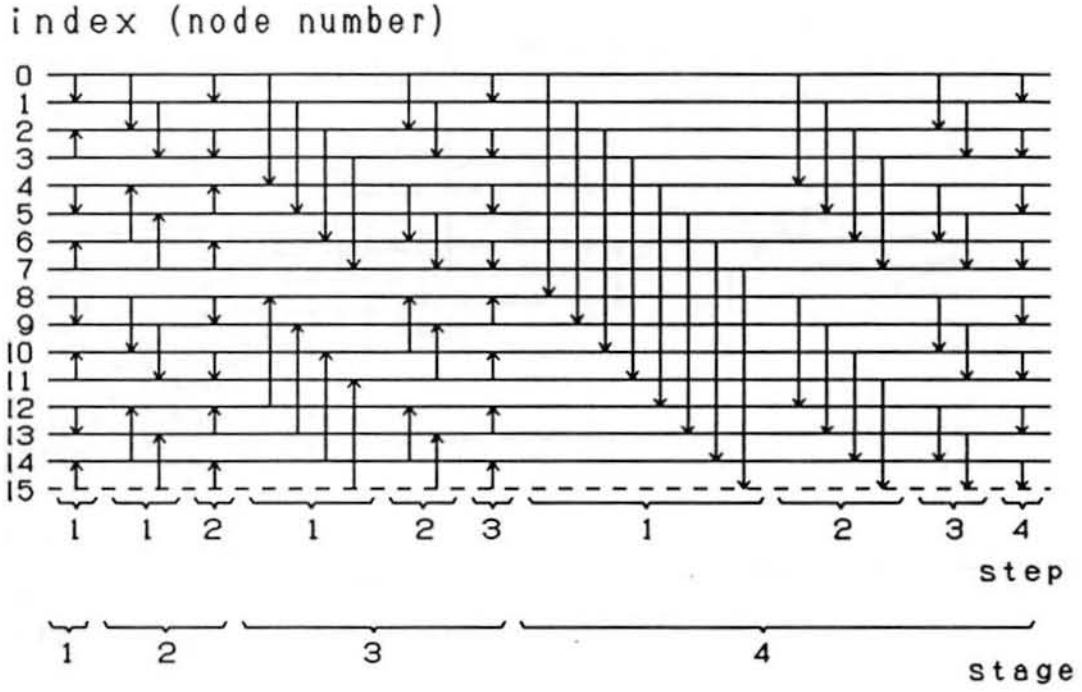


図4．12　$n-1$ 番目のノード障害に適用できるバイトニック・ソート

Fig.4.12 Bitonic sort for the n-1th node failure.

4.4.1 証明

2.2.1 で述べたバイトニック・ソートの原理において、2^{i+1} 個のデータの双調列から2組の 2^i 個のデータの双調列を生成する際バイトニック・ソートの性質により、大きい方の 2^i 個のデータの双調列と小さい方の 2^i 個のデータの双調列とに分かれる。ノード障害が無い場合には、そのまま再帰的に $2^{i-2}, 2^{i-3}\ldots$ と短い双調列を作っていけば最後に単調列が完成する。

ところが、ノード障害が有る場合には、再帰性を単純に使うことができない。2^{i+1} 個のデータの双調列から2組の 2^i 個のデータの双調列に分ける際、そのままでは一般に一方の組の各値が他方の組の各値を超えないような関係を維持することができず、冗長ステップやインデックス距離 x の変更が必要となる。

しかし、図4．11のタイプにおいては、先頭の0番目のノードが障害を起こして使えない場合には、そのままでもこの関係を維持することができる。

以下に、その証明を示す。

双調列 $(a_0, a_1, a_2, \cdots, a_{m-1}, a_m, \cdots, a_{2m-1})$ を、

（a）$(a_0, a_1, a_2, \cdots, a_{m-1})$

（b）$(a_m, a_{m+1}, a_{m+2}, \cdots, a_{2m-1})$

と2分割して考える。

データ a_0 のあるノード0は障害ノードとし、使用不能と考える。又、双調列の最大値はノード i にあり値を a_i とする。

（1） a_i が（a）側にある時、

$$a_0 \le a_1 \le \cdots \le a_i \ge a_{i+1} \ge \cdots \ge a_{m-1} \qquad \text{（a）}$$
$$\ge a_m \ge a_{m+1} \ge \cdots\cdots \ge a_{2m-2} \ge a_{2m-1} \qquad \text{（b）}$$

任意のノード $x\ (0 \le x \le m-1)$ の値 a_x は、ノード $m+x$ の値 a_{m+x} と比較が行われ、双調列の性質により、（a）側の値は（b）側の値を超えないような関係になる。ところが、ノード m の値 a_m は、比較の対象となるノード0（値 a_0）が障害ノードのため、a_0 と比較されずにそのまま（b）側に残る。

一方、（b）の関係、

$$a_{2m-1} \le a_{2m-2} \le \cdots \le a_{m+1} \le a_m$$

より、a_m は昇順に数えて少なくとも m 番目以上の値である。

（a）側は、ノード0が障害ノードのため、使用可能なノードは $m-1$ 個、（b）側は m 個のため、a_m が（b）側にそのまま残っても、（a）側の値は全て、（b）側の値を超えることがないことが保障される。

（2） a_i が（b）側にある時、

$$a_0 \le a_1 \le a_2 \le \cdots\cdots\cdots \le a_{m-1} \qquad \text{（a）}$$
$$\le a_m \le \cdots \le a_i \ge a_{i+1} \cdots \ge a_{2m-1} \qquad \text{（b）}$$
$$a_0 \le a_1 \le a_2 \le \cdots\cdots \le a_{m-1} \le a_m$$

より、a_m は a_0 を除き昇順に数えて少なくとも m 番目以上の値である。

従って a_m が（ｂ）側にそのまま残っても、（ａ）側の値は全て、（ｂ）側の値を超えることがないことが保障される。

図４．１２のタイプのアルゴリズムの場合も a_i を最小値として、同様に証明できる。

4.4.2 有弦環結合ネットワークへの応用

ここに示した特殊なケースにおいては、ノード障害時にバイトニック・ソートを行う場合に、冗長ステップを設ける必要もないし、各ステップにおいてインデックス距離 x を変更する必要もない。

この特徴を生かして、障害ノードが検出されたならば、図４．１３のようにその右隣のノードから時計回りにインデックスを $0,1,2,\cdots,n-1$ とつけて行き、図４．１２のタイプのアルゴリズムを適用してソーティングを行うようにすれば、有弦環結合の任意のノード障害に対応できる。

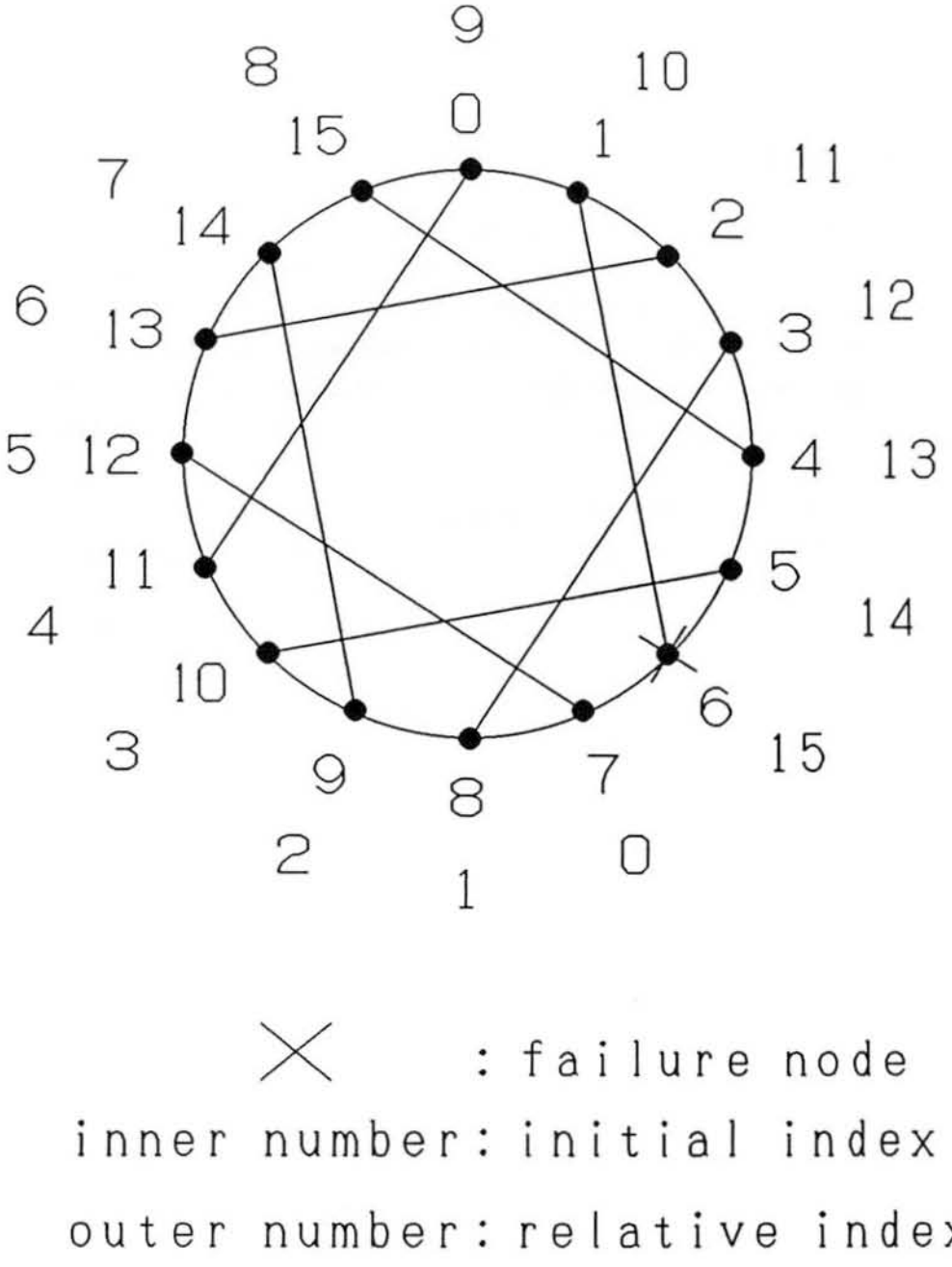


図4．13　有弦環結合と4．4節での手法の適用方法
Fig.4.13 Chordal ring connection and
the method of section4.4 applied on it.

5 結論

初めに有弦環結合ネットワーク型超並列コンピュータにおいて、バイトニック・ソートを用いて並列にソーティングを行う方法について述べ、ノード数、データ数 n の場合の処理時間が $O(\sqrt{n}\log n)$ であることを示した。

次に、リンクが障害を起こした場合の代替経路のとり方について示した。

障害箇所については、任意の1本のリンクの場合と2本のリンクの場合についてとり上げた。また、有弦環結合においては、各リンクの代替経路が常に5となることを示し、バイトニック・ソートでこの手法を用いた場合の、リンク障害時の全経路長の評価を行った。

3番目に、ノード障害時に冗長ステップを加え、処理の一部を変更することにより、障害ノードを除外してバイトニック・ソートを行う手法を示した。また、特殊なケースとして、冗長ステップの追加や処理の変更が不要な場合について述べ、有弦環結合ネットワーク上への適用方法を示した。

今後は、他の形状のネットワークでの検討やソーティング以外の処理についての検討が必要と考えられる。

謝辞

本研究にあたり御指導いただきました山浦弘夫教授、有益なアドバイスをいただきました山本博章助教授に深く感謝いたします。

また、本研究に入るきっかけを与えて下さいました防衛大学校情報工学教室 古賀義亮教授に深く感謝いたします。

中野康明教授、中村八束教授、岡本正行助教授におかれましては、本論文を通読していただくとともに有益なご助言、討論を賜り厚くお礼申し上げます。

さらに、大学院生 船木英岳氏には本論文作成にあたりご協力いただき深く感謝いたします。

研究成果の発表

学会誌

（１） 小林洋、古賀義亮：“有弦環結合ネットワーク型コンピュータによるソーティング”，電子通信学会論文誌（Ｄ），J68-D, 3, pp.253-260(1985-03).

（２） 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”，電子情報通信学会論文誌（Ｄ－Ｉ），J75-D-I,5, pp.280-287(1992-05).

（３） 小林洋、山本博章、山浦弘夫、船木英岳：“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”，電子情報通信学会論文誌（Ｄ－Ｉ），J76-D-I,9,pp.465-472(1993-09).

（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”，電子情報通信学会論文誌（Ｄ－Ｉ），J75-D-I,10,pp958-961(1992-10).

（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の改良法”，電子情報通信学会論文誌（Ｄ－Ｉ），採録決定

講演

（１） 小林洋、船木英岳、山本博章、山浦弘夫：“有弦環結合ネットワーク型コンピュータ上でのノード障害時の代替経路”，電気関係学会東海支部連合大会講演論文集，p378(1993-10).

参考文献

（１） 小池誠彦：“超並列マシン”，情報処理，**28**，1，pp.94-105 (1987-01).

（２） 馬場敬信：“超並列マシンへの道”，情報処理，**32**，4，pp.348-364 (1991-04).

（３） 黒川恭一、相磯秀夫：“並列処理の諸問題：結合方式”，情報処理，**27**，9，pp.1005-1021(1986-09).

（４） 高橋義造：“並列処理のためのプロセッサ結合方式”，情報処理，**23**,3,pp.201-209(1982-03).

（５） ミード，コンウェイ著，菅野，榊訳：“超ＬＳＩシステム入門”，培風館，pp.291-359(昭５６－０６).

（６） Arden,B.W. and Lee,H.；“*Analysis of Chordal Ring Network*”，IEEE Trans. Comput., **C-30**, 4, pp.291-295(1981).

（７） Thompson,C.D. and Kung,H.T.：“*Sorting on a Mesh－Connected Parallel Computer*”，Commu.ACM, **20**, 4, pp.263-271(1977).

（８） Nassimi,D and Sahni,S.；“*Bitonic Sort on a Mesh－Connected Parallel Computer*”，IEEE Trans. Comput., **C-27**, 1, pp.2-7(1979).

（９） Stone,H.S.；“*Parallel Processing with the Perfect Shuffle*”，IEEE Trans. Comput., **C-20**, 2, pp.153-161(1971).

（１０） Preparata,F.P. and Vuillemin,J.：“*The Cube－Connected Cycles : A Versatile Network for Parallel Computation*”，Commu.ACM, **24**, 5, pp.300-309(1981).

（１１） Baudet,G. and Stevenson,D.；“*Optimal Sorting Algorithms for Parallel Computers*”，IEEE Trans. Comput., **C-27**, 1, pp.84-87(1978).

（１２） 田中譲：“データベース処理や文書処理を高速化するサーチ／ソフト・ハードウェアの動向”，日経エレクトロニクス，pp.141-178(1983-8-1).

（１３） Batcher,K.E.：“*Sorting networks and their applications*”,Proc.AFIPS 1968 SJCC, **32**, AFIPS Press, pp.307-314(1968).

（１４） Knuth,D.E.：“*The Art of Computer Programming, Vol.3 : Sorting and Searching*”，Addison-Wesley, Reading, Mass., pp.220-246(1973).

A1. バイトニック・ソートの証明[13]

双調列 $(a_1, a_2, \cdots, a_{2n})$ において、$d_i = \min(a_i, a_{n+i}), e_i = \max(a_i, a_{n+i})$ とすると、

$$\max(d_1, d_2, \cdots, d_n) \leq \min(e_1, e_2, \cdots, e_n)(1 \leq i \leq n)$$

となることを証明する。

つまり、双調列を半分に分け $(a_1, a_2, \cdots, a_n), (a_{n+1}, a_{n+2}, \cdots, a_{2n})$ とし、それぞれの i 番目の max と min からなる順序列を作ると、すべての min 側の値 $(d_1, d_2, \cdots, d_n)$ は、全ての max 側の値 $(e_1, e_2, \cdots, e_n)$ を超えることがない事を証明する。

(証明)

双調列 $a_1 \leq a_2 \leq a_3 \leq \cdots \leq a_{j-1} \leq a_j \geq a_{j+1} \geq \cdots \geq a_{2n}(1 \leq j \leq 2n)$ において、$n < j \leq 2n$ のとき、($1 \leq j \leq n$ のときは、順序を反対にすれば良い。)

(1) $a_n \leq a_{2n}$のとき、

$a_1 \leq a_2 \leq a_3 \leq \cdots \leq a_n \leq a_{n+1} \leq \cdots \leq a_j \geq \cdots \geq a_{2n}$

であるから、

$$\underbrace{a_1 \leq a_2 \leq a_3 \leq \cdots \leq a_n}_{\text{min 側 } n \text{ 個}} \leq \underbrace{a_{n+1} \leq \cdots \leq a_j \geq \cdots \geq a_{2n}}_{\text{max 側 } n \text{ 個}}$$

$$\begin{cases} d_i = a_i \\ e_i = a_{n+i} \end{cases} (1 \leq i \leq n)$$

よって (全ての d_i) $\leq$ (全ての e_i)

(2) $a_n > a_{2n}$ のとき、

$$\left\{ \begin{array}{lcl} a_{k-n} & \leq & a_k \\ a_{k-n+1} & > & a_{k+1} \end{array} \right\} \text{なる } k\text{が存在する}$$

$a_1 \leq a_2 \leq \cdots \leq a_{j-n} \leq \cdots \leq a_{k-n} \leq a_{k-n+1} \leq \cdots \leq a_n \leq \cdots \leq a_j \geq \cdots \geq a_k \geq a_{k+1} \geq \cdots \geq a_{2n}$

a. $1 \leq i \leq k-n$ のとき、

$$\begin{cases} d_i = a_i \\ e_i = a_{n+i} \end{cases}$$

(a) $1 \leq i < j-n$ のとき、昇べき順の範囲内であり、$a_{n+(j-n)} = a_n$ であるので、

$$\begin{cases} d_i & \leq & d_{i+1} \\ e_i & \leq & e_{i+1}, e_n \leq e_1 \end{cases}$$

明らかに、

$$\underbrace{a_1 \leq a_2 \leq \cdots \leq a_{j-n-1}}_{\text{min 側}} \underbrace{\leq a_{n+1} \leq \cdots \leq a_{j-1}}_{\text{max 側}}$$

(b) $j-n \leq i \leq k-n$ のとき、

$$\begin{cases} d_i & \leq & d_{i+1} \\ e_i & \geq & e_{i+1} \end{cases}$$

$$\underbrace{a_{j-n} \leq \cdots \leq a_{k-n}}_{\text{min 側}} \underbrace{\leq a_k \leq \cdots \leq a_j}_{\text{max 側}}$$

b．$k-n+1 \leq i \leq n$ のとき、

$$\begin{cases} d_i & = & a_{n+i} \\ e_i & = & a_i \end{cases}$$

$$\underbrace{a_n > \cdots > a_{k-n+1}}_{\text{max 側}} \underbrace{> a_{k+1} > \cdots > a_{2n}}_{\text{min 側}}$$

よって、(全ての d_i) $\leq$ (全ての e_i) □

A2. 有弦環結合ネットワークの弦の長さの最適値について(6)

n 個のノードからなる有弦環結合ネットワークの弦の長さの最適値 w_{opt} を、本論文では w_{opt} は $\sqrt{n}$ に近い奇数の値 (但し、$n=4$ のときは $w_{opt}=2$ 、$n=8$ のときは $w_{opt}=4$) とし、この時ノード間の最大距離 (直径) d は最小となり $d \doteq \sqrt{n}$ となるとしたが、厳密にいうとこの弦の長さの値は最適値ではない。定理によると、直径 d が与えられた時、ネットワークのノードの最大数 n_{max} と弦の長さの最適値 w_{opt} は次のようになる。

(1) $d=2$ のとき、
$n_{max}=6, w_{opt}=3$

(2) $d=3$ のとき、
$n_{max}=14, w_{opt}=5$

(3) $d=4$ 又は 6 のとき、
$n_{max}=d^2+2d-4, w_{opt}=d+3$(及び $w_{opt}=d+1$, 但し $d=4$ のときのみ)

(4) d が奇数で、$d \geq 5$ のとき
$n_{max}=d^2+3d-6, w_{opt}=d+4$(及び $w_{opt}=d+2$, 但し $d=5$ のときのみ)

(5) d が偶数で、$d \geq 8$ のとき、
$n_{max}=d^2+3d-12, w_{opt}=d+5$(及び $w_{opt}=d+3$, 但し $d=8$ のときのみ)

さらに厳密にいうと、演算の種類によって、使用するノード間の頻度 (重み) が異なるので、このことを考慮して、線形計画法等で求めるべきであろう。

A3. 正方格子 (Mesh) 結合ネットワークでの処理時間

正方格子ネットワークでインデックスを図 A のようにつける。(このようなインデックスのつけ方を Shuffled row major indexing[7] と言う。)

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

図 A

ノード数を $n = 2^{2i}$ とすると、全経路時間 T_R は次のようにして求まる。(以下単位時間 t_r 省略)

i が偶数の時の経路時間 t_{Re} と奇数の時の経路時間 t_{Ro} は、次式のようになる。

$$\begin{aligned} t_{Re} &= 2 \times (2^1 + 2^2 + \cdots + 2^i) = 4 \cdot 2^i - 4 && \text{(A7)} \\ t_{Ro} &= (2^1 + 2^2 + \cdots + 2^i) + (2^1 + 2^2 + \cdots + 2^{i-1}) \\ &= 3 \cdot 2^i - 4 && \text{(A8)} \end{aligned}$$

従って、各々の場合の全経路時間 T_{Re}, T_{Ro} は次式のようになる。

$$\begin{aligned} T_{Re} &= \sum t_{Re} = 4 \times (2^1 + 2^2 + \cdots + 2^i) - \underbrace{(4 + 4 + \cdots + 4)}_{i \text{ 個}} \\ &= 8 \cdot (2^i - 1) - 4i && \text{(A9)} \\ T_{Ro} &= \sum t_{Ro} = 3 \times (2^1 + 2^2 + \cdots + 2^i) - \underbrace{(4 + 4 + \cdots + 4)}_{i \text{ 個}} \\ &= 6 \cdot (2^i - 1) - 4i && \text{(A10)} \end{aligned}$$

従って、

$$T_R = T_{Re} + T_{Ro} = 14 \cdot (2^i - 1) - 8i \qquad \text{(A11)}$$

となる。

$n = 2^{2i}$ より $2^i = \sqrt{n}, i = \frac{1}{2}\log n$ であるので、

$$T_R = 14(\sqrt{n} - 1) - 4\log n \qquad \text{(A12)}$$

となる。