相互結合網 RDT における Adaptive routing

舟橋 啓† 塙 敏博† 工藤知宏‡ 天野英晴†

†慶應義塾大学理工学部
‡東京工科大学 情報工学科

相互結合網 RDT は再帰構造を持つ Torus の重なりから構成されており, 超並列計算機のプロセッサ間接続を行なうために優れた性質を多く持っている. 本研究では,Duato により提案された方法と Turn モデルを利用した方法の 2 つのアプローチにより RDT 上でデッドロックフリーな adaptive routing を実現する方法を示す.

Duato による必要十分条件を用いて,PRDT 用の adaptive routing を 2 種類提案した. これらの方法は, ネットワーク中の利用されていないチャネルを最大限に活用することが出来, 高い通過率を示すことが予想される.

更に,Turn モデルを用いた adaptive routing を提案した. このルーティングは PRDT, RDT(2,4,1)/α のどちらにも使用することができる. また, ランクの使用順は自由, 最短経路を通る必要がない, 各次元内で逆方向に方向転換が可能という点で Duato の方法よりも高い自由度を示す.

相互結合網, 適応型ルーティング, デッドロック回避,RDT

Adaptive routing on the Recursive Diagonal Torus

A. Funahashi† T.Hanawa† T.Kudoh‡ H. Amano†

†Keio University
‡Tokyo Engineering Univesity

Recursive Diagonal Torus, or RDT consisting of recursively structured tori is an interconnection network for massively parallel computers. In this paper, an adaptive routing on the RDT is proposed by using a necessary and sufficient confition for deadlock-free adaptive routing proposed by Duato.

By using Duato's necessary and sufficient condition, we proposed two adaptive routing algorithms on the RDT. Since channels are used efficiently with these algorithms, the performance can be improved.

We also proposed a new adaptive routing algorithm by using Turn model. This routing algorithm can be applied on both the PRDT and RDT(2,4,1)/α. Also by using this algorithm, ranks can be used in the free turn and it doesn't have to use the minimal path. It can exploit higher flexibility than the above two algorithms since it can use vectors in the reverse direction.

Interconnection Network, Adaptive Routing, Deadlock Avoidance, RDT

1 はじめに

RDT(Recursive Diagonal Torus)[7]は, 比較的小さいdegreeで小さな直径と高いランダム転送能力を実現する相互結合網であり, 文部省重点領域研究で開発が進んでいる超並列計算機テストベッドJUMP-1[5]用にルータチップが開発されている[10].

RDTは, ツリー, ハイパーキューブのエミュレーションが容易である等, 超並列網として優れた特徴を持つ一方, メッシュを内蔵していることから現在のメッシュ/トーラス結合並列計算機上で開発されたアルゴリズムの移植が容易である. さらに, ルートを複数持つ階層構造を利用して分散共有メモリを実現する方法の検討が行なわれている[9][12].

RDTのルーティングは, 出発地から目的地までの経路のベクトルを分解するベクトルルーティングを基本とし[7], e-cubeルーティング[1]を利用して, デッドロックを回避する方法が提案され[8], 実際にルータチップでも使われている. しかし, この方法は, デッドロックを防ぐために, ランクを使用する順序およびベクトルを適用する順序が決っているため, 効率が悪い上, 混雑や故障の迂回を行なうことができない. 一方, 経路の選択に自由度を持つルーティング法としてFloating Vector Routing[8]も提案されているが, この方法ではデッドロックの可能性がある.

そこで, 本研究では, 近年, 急速に研究が進んだadaptive routingをRDTに適用し, 効率が良く, 故障や混雑の迂回が可能なルーティング法を提案する.

2 RDTの構成

RDTの定義と基本的なルーティング法について, 本論文での議論に必要な範囲を以下にまとめる. 詳細な定義については[11]を参照されたい.

RDTは, 二次元トーラスを基本とする. 簡単のため基本トーラスは $N_{x0} \times N_{y0}$ ($N_{x0} = N_{y0} = c(2n)^{\gamma}$, c, n, γ は正の整数)であるとし以下のように番号付けられているとする.

定義 1 : 基本トーラス

(0,0)	(1,0)	(2,0)	$\cdots$	(N_{x0}-1,0)
(0,1)	(1,1)	(2,1)	$\cdots$	(N_{x0}-1,1)
(0,2)	(1,2)	(2,2)	$\cdots$	(N_{x0}-1,2)
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
(0,N_{y0}-1)	(1,N_{y0}-1)	(2,N_{y0}-1)	$\cdots$	(N_{x0}-1,N_{y0}-1)

上に示すノードの二次元アレイ上で, ノード (x, y) が四方の隣接ノード *(node (mod($x \pm 1, N_{x0}$), y) (x, mod* $(y \pm 1, N_{y0})$ *))* との間に結合リンクを持つ構造を基本トーラスまたはランク *0* トーラスと呼ぶ. □

一般に, トーラス構造に対してバイパスリンクを定める場合, 最も効果的なのは対角線方向である. 今, 各ノード (x, y) が, $(x \pm n, y \pm n)$ と結ぶ4本の付加リンクを持つとすると, この付加リンクは新たなトーラス状の結合網を形成する. このトーラス状結合網は, 基本トーラスに対し45度傾き, グリッドサイズは$\sqrt{2}$n倍になっている.

このトーラス状結合網をランク1トーラスと呼び, 正の整数nを基数と呼ぶ. さらに, ランク1トーラス上に同様な方法で付加リンクを付け加え, ランク2トーラスを形成する. 以上の操作を再帰的に繰り返し, 次々とランクの高いトーラスを形成していく.

我々の提案したRDT(Recursive Diagonal Torus)は, 以上の方法により再帰的に構成したトーラス状結合網の組合せにより構成される. ここで, 偶数ランクのトーラスが基本トーラスと同様の2次元正方隣接格子になるのに対し, 奇数ランクのトーラスは縦横比が2:1で循環ループが螺旋状になる. 一般的に, トーラスは, 基本トーラス同様の正方隣接格子で単純な循環構造を持つ結合網を指すが, ここでは, 螺旋状の循環ループを持つ奇数ランクの結合網も一括して定義し, 共に区別なく「トーラス」と呼ぶことにする.

定義 2 : 上位トーラス
ランク r トーラスが以下の式を満足するサイズを持つとする.

$$N_{x(r+1)} = \frac{N_{yr}}{gcd(N_{xr}, n)} > 1$$

$$N_{y(r+1)} = \frac{N_{xr}}{2gcd(N_{yr}, n)} > 1$$

このランク r トーラス上の任意のノード (x, y) と, 以下に示すノード (x', y') と (x'', y'') との間にリンクを付加することにより構成することができる結合網をランク $r+1$ トーラス *(*サイズは $N_{x(r+1)} \times N_{y(r+1)}$*)* と呼ぶ.

$$x' = (x+n) - (N_{xr} - N_{yr})\lfloor \frac{x+y}{N_{xr}-2n} \rfloor - N_{xr}\lfloor \frac{x+n}{N_{xr}} \rfloor \lfloor \frac{N_{yr}}{N_{xr}} \rfloor$$

$$y' = (y+n) - (N_{xr} - N_{yr})\lfloor \frac{x+y}{N_{xr}-2n} \rfloor - N_{yr}\lfloor \frac{y+n}{N_{yr}} \rfloor \lfloor \frac{N_{yr}}{N_{xr}} \rfloor$$

$$x'' = (x+n) - (N_{xr} - N_{yr})\lfloor \frac{x-y}{N_{xr}-2n} \rfloor - N_{xr}\lfloor \frac{x+n}{N_{xr}} \rfloor \lfloor \frac{N_{yr}}{N_{xr}} \rfloor$$

$$y'' = (y-n) + (N_{xr} - N_{yr})\lfloor \frac{x-y}{N_{xr}-2n} \rfloor - N_{yr}\lfloor \frac{y-n}{N_{yr}} \rfloor \lfloor \frac{N_{yr}}{N_{xr}} \rfloor$$

ここで, gcd は最大公約数を示す. ランク $r+1$ トーラスのノード番号は以下のように定める.

1. 座標軸の原点を, ランク r トーラスの原点のノードに置く.

2. 座標軸の方向は, r が偶数の場合は基本トーラスと同一とし, 奇数の場合は基本トーラスに対し時計回りに 45 度傾ける.

3. ランク $r+1$ トーラスを形成する各ノードに対し, 座標軸に従って基本トーラスと同一方向に二次元番号 (x_{r+1}, y_{r+1}) を与える. □

定義 1に示した基本トーラスに対し, 定義 2 を再帰的に用いて上位トーラスを形成することにより, RDT (Recursive Diagonal Torus) を定義する.

定義 3 : 完全 RDT
ランク 0 トーラス (基本トーラス) 上に定義 2に基づき再帰的にトーラスを形成したとき, 全てのノードが, 形成可能な全ての上位トーラスを形成するためのリンクを持つ結合網を完全 *RDT* (**PRDT(n,R)**) と呼ぶ. ここで, n は基数, R は最大ランク数である. □

完全 RDT は, degree が $(4(R+1))$ で大き過ぎることから非現実的な結合網であるが, アルゴリズム構築上重要である. RDT のルーティング アルゴリズム, ブロードキャスト, 他の網のエミュレーションは全て完全 RDT を念頭において考え, 実際の RDT に対して写像する.

定義 4 : RDT
ランク 0 トーラス (基本トーラス) 上に定義 2 に基づき再帰的にトーラスを形成したとき, 各ノードが, 基本トーラス及び基本トーラスをランク 0 として順に形成した他の m 個の上位トーラスを形成するためのリンクを持つ結合網を **RDT(n,R,m)** と呼ぶ. ここで, n は基数, R は最大ランク数である. また, m を多重度と呼ぶ. □

以上の定義では, 各ノードは他のノードと違ったランクの上位トーラスを持つことができる点に注意されたい. 以下, ノードがあるトーラスを形成するために必要なリンクを持つことを, 「トーラスを持つ」と表現する. RDT(n,R,m) は基本トーラスと m 個の上位ランクのトーラスを持つため, degree は 4(m+1) となる.

実際に 1 万ノード程度のサイズを念頭においた場合, n=2 , m=1 に取るのが有利であり, 現在,JUMP-1 用に図 1に示す構成の RDT, RDT(2,4,1)/αについて検討を進めている.

n=2 の場合, あるランク上には独立なトーラス数が 8 つ形成される. RDT(2,4,1)/αでは, 基本トーラス上に形成される 8 つのランク 1 トーラスのうち, 2 つをそのままランク 1 トーラスを形成するのに用い, ランク 2, 3, 4 を形成するために, それぞれ 2 つずつを用いる. すなわち, ランク 1 から 4 まで各ランクのトーラスを持つノードの数は等しい.

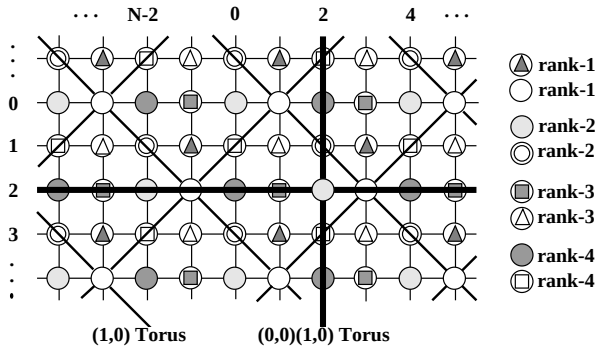

図 1: RDT(2,4,1)/αの構成

この構成では図 1に示すように, あるノードが持っていない上位トーラスは隣接ノードのどれかが持っている. すなわち, 基本トーラス (ランク 0-トーラス) 上の 1 ステップの移動で全ての上位トーラスが利用可能である.

3 RDT 上でのルーティングアルゴリズム

3.1 ベクトルルーティング

ベクトルルーティングは, 出発値から目的値までの経路を, 各ランクのトーラスの一辺を単位ベクトルとするベクトルの合成によって表現することにより, 使用するトーラスのランクとリンクの方向を求める方法である. 今, 出発値ノードから目的地ノードまでのベクトルは, 基本トーラスの x 方向, y 方向の単位ベクトルを $\vec{x_0}$, $\vec{y_0}$ とすると, $\vec{A} = \alpha\vec{x_0} + \beta\vec{y_0}$として表される. ここで$\alpha$ と βは, ノード番号の差により, 簡単に求められる. さて, このベクトル$\vec{A}$を図 2(a) に示すように, 上位トーラスの単位ベクトルの合成の形に変換していく. まず, 各上位トーラスの単位ベクトルの方向を図 2(b) のように, 45 度ずつ時計回りに巡回するように定める. ここで, ランク $i+1$ トーラスの単位ベクトルは, ランク i の単位ベクトルを用いて, 以下のように表すことができる.

$$\vec{x}_{r+1} = n\vec{y_r} + n\vec{x_r} \quad (1)$$

$$\vec{y}_{r+1} = n\vec{y_r} - n\vec{x_r} \quad (2)$$

ここで, $\vec{x}_{r+1}, \vec{y}_{r+1}, \vec{x_r}, \vec{y_r}$はそれぞれのランクの単位ベクトルで, n は基数である. この二式より, ランク r に存在するベクトル$\vec{A_r} = \alpha_r\vec{x_r} + \beta_r\vec{y_r}$は

$$\alpha\vec{x_r} + \beta\vec{y_r} = \alpha_{r+1}\vec{x}_{r+1} + \beta_{r+1}\vec{y}_{r+1} + a_r\vec{x_r} + b_r\vec{y_r} \quad (3)$$

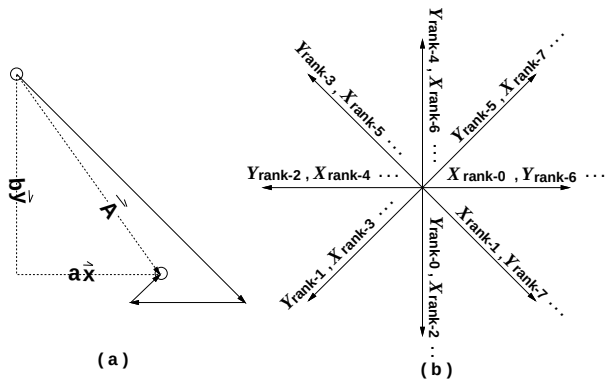

図 2: ベクトル分解と回転座標

に変換される. 式 (1) にα_{r+1}を, 式 (2) にβ_{r+1}を掛けて, 式 (3) に代入すると

$$\alpha_r\vec{x_r}+\beta_r\vec{y_r}=\alpha_{r+1}n\vec{x_r}+\alpha_{r+1}n\vec{y_r}+\beta_{r+1}n\vec{y_r}-\beta_{r+1}n\vec{x_r}$$

単位ベクトルの方向について以下の式が成立する.

$$\alpha_r\vec{x_r}=\alpha_{r+1}n\vec{x_r}-\beta_{r+1}n\vec{x_r} \quad (4)$$

$$\beta_r\vec{y_r}=\alpha_{r+1}n\vec{y_r}+\beta_{r+1}n\vec{y_r} \quad (5)$$

式 (4) と (5) を解き, α_{r+1}とβ_{r+1}を求めると,

$$\alpha_{r+1}=\frac{\beta_r+\alpha_r}{2n} \quad (6)$$

$$\beta_{r+1}=\frac{\beta_r-\alpha_r}{2n} \quad (7)$$

となる. ここで, α_{r+1}とβ_{r+1}は整数でなければならないため, 式 (6),(7) に最も近いの整数α'_{r+1}とβ'_{r+1}を選ぶ. 割算の値は n 捨 $n+1$ 入される. 例えば, $n=2$ では 2 捨 3 入, $n=3$ では 3 捨 4 入である.

このα'_{r+1}, β'_{r+1}を式 (3) のα_{r+1}, β_{r+1}にそれぞれ代入すると, a_rと b_rは以下の式で表される.

$$a_r=\alpha_r-n\alpha'_{r+1}+n\beta'_{r+1} \quad (8)$$

$$b_r=\beta_r-n\alpha'_{r+1}-n\beta'_{r+1} \quad (9)$$

ここで, α'_{r+1}とβ'_{r+1}から成るベクトルに関して同様に分解を繰り返せば, 下から順に利用すべきトーラスとそのリンク（ベクトルの方向より）を知ることができる.

具体的なアルゴリズムを C 言語のプログラム片の形で示す.

```
for (rank=0; rank<MAX_RANK; rank++) {
    g = div_2n(a+b); f= div_2n( -(a-b));
    vector[rank].x= a-n*(g-f);
    vector[rank].y= b-n*(g+f);
    a=g ; b= f;
}
vector[MAX_RANK].x=g; vector[MAX_RANK].y=f;
```

ここで, 関数 div_2n は, 2n での割算を示すが, 答えは n 捨 $n+1$ 入される. またプログラム片中の a, b, g, f, j, k は式 (1)-(9) に示すα_r, β_r, α'_{r+1}, β'_{r+1}, a_r, b_r にそれぞれ対応する.

(1,2) から (5,9) へのルーティングのベクトル分解を $n=2$ とした場合について図 3に示す. ベクトルルーティング は操作が簡単で, 利用するベクトルの順番を変えることにより, 代替経路が得られるため, 混雑や故障箇所の迂回が可能である. しかし, ベクトルルーティング は deterministic routing のため, パケットがノードを出発するときに既にルーティングは決定している. そのため, 各ノードが結合網中の全ての状況を把握していなければならない.

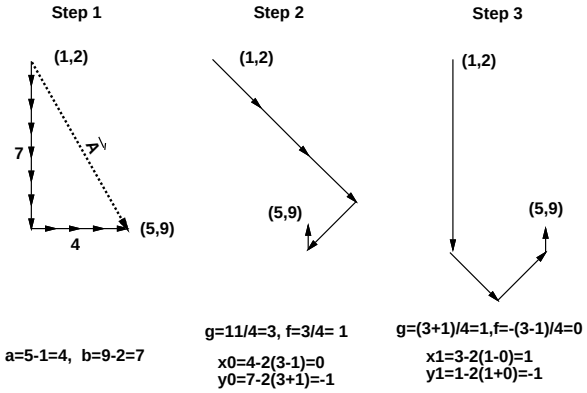

図 3: ベクトル ルーティングの例

3.2 RDT 上での adaptive routing

deterministic routing に対し,adaptive routing は, 経路を動的に選んでルーティングするため, 混雑や故障を発見したときにその場で回避することが可能である. また, 空いている経路を有効に用いることで, 結合網の性能を最大限に引き出すことができ, 局所的な混雑を回避する点においても有効である. 耐故障性の点でも, 各ノードが送信しようとした経路が故障していたら, その場で他の経路を選択できるため deterministic routing に対し有利である.

しかし, たとえ adaptive routing が行えたとしても, デッドロックを起こす可能性がある. そこで,2 つのアプローチから RDT 上でのデッドロックフリーな adaptive

routing を提案する. 基本的にはベクトルルーティングで求めたベクトルの使用する順序を変更することにより代替経路を求めるが, 最短経路を使用する必要がない場合は, 新たにベクトルを利用していくことも可能である.

4 Duatoの必要十分条件による方法

4.1 k-ary n-cube

今まで提案されてきたデッドロックフリーな adaptive routing は, 何らかの方法で循環を断ち切ることによって, デッドロックを起こさない様にしてきた. しかし Duato は, 循環を含む経路に対してもデッドロックを起こさない adaptive routing の必要十分条件を示した [2].

Duato の条件は,

a. 結合網全体に渡る循環の無い逃げ道 (escape path) を用意する

b. 逃げ道と, 逃げ道により循環が切断されてデッドロックが起きなくなる他の経路によって結合網中のどのノード間でもパケットが送れるようにする

の 2 点を満足できれば a.,b. の経路を adaptive routing で選択することにより, デッドロックしない adaptive routing が可能になる.

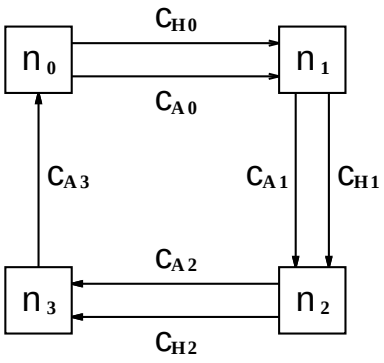

図 4: デッドロックフリーの証明

Duato は, 以下の手順により,k-ary n-cube での adaptive routing を実現した.

1. はじめに単方向リング (図 4) のネットワークについて考える. パケットが存在しているノードより目的地のノード番号が大きい場合は, チャネル C_H を使い, 低い場合は C_A を使用することで escape path が保証される.escape path が全てのノード間に存在するため, このネットワークはデッドロックフリーである.

2. 次に, ネットワークを双方向リングに拡張する. しかし,Duato の方法では, ルーティングは常に最短経路を通る (minimal routing) から, ルーティングの途中でそれまで使用していたリンクと逆方向のリンクを使用することはありえず, 逆方向のリンクは全く別のネットワークとして分離して考えることができる. よって, 双方向リングのネットワークでもデッドロックフリーである.

3. 次に, ネットワークを多次元のものに拡張する. e-cube routing[1] と同様に, 次元の使用順を固定すると, 結局は双方向 (単方向) リングを決まった順に使用するだけになる. よって, ルーティングのときに使用されるチャネル間の依存関係は各次元で独立であるため, デッドロックフリーである.

   以上により,escape path C_1 が用意された.

4. ここで, バーチャルチャネル $C_F(Fully\ adaptive)$ を新たに用意し, そのチャネルでは次元の使用順を無視して移動可能とする. これにより, デッドロックフリー な adaptive routing が可能となる.

ただしこのルーティングでは,2. を満足させるために, 常に最短経路を通るという,minimal routing を守らなければならない.

4.2 PRDT

上に述べた k-ary n-cube での adaptive routing を PRDT 用に拡張する.

1. k-ary n-cube での 3. までと同じ手順で escape path C_1 を用意する. この escape path は, 単一トーラスでのもので, ランク間の移動を想定していない.

2. 次に, ランクの使用順を固定する. 数字をランク, X or Y を次元とすると,
   $Xrank3 \rightarrow Yrank3 \rightarrow Xrank2 \rightarrow Yrank2 \rightarrow Xrank1 \rightarrow Yrank1$
   (ランク 3 の X 方向から順に Y 方向, ランク 2 の X 方向$\cdots$) という順に channel を使う.e-cube routing と同様の考え方により, このチャネル集合は, デッドロックフリーである. これにより PRDT 用の escape path C_1' が用意された.

3. ここで,adaptive routing にするには二つの方法が考えられる.

   3.a バーチャルチャネル C_F を C_1' 全体に用意する (図 5). 新たに用意されたバーチャルチャネ

ルでは,
$Xrank3 \rightarrow Yrank3 \rightarrow Xrank2 \cdots$
の順を無視してチャネルを使用可能とする.
$Xrank3, Yrank3, Xrank2, \cdots$ は一般的なトーラスの次元が増えたものと考えられるため, 同様にデッドロックフリーな adaptive routing が可能となる.

ただし,Duato の条件を利用するには minimal routing を用いる必要があり, ルーティングの途中で進んできた方向と逆の方向に向かうことは許されない. つまり RDT では, 偶数ランク, 奇数ランク内での $+X, +Y$ 方向は共通でなければならない. ベクトルルーティングでは, 各ランクでの単位ベクトルは式 (1),(2) であらわせるように,45 度ずつ回転していくが, 偶数ランクでの $+X, +Y$ 方向はベーストーラスでの単位ベクトル $\vec{x}_0, \vec{y}_0$ とし, 奇数ランクでの $+X, +Y$ 方向はランク 1 トーラスでの単位ベクトル $\vec{x}_1, \vec{y}_1$ としなければならない.

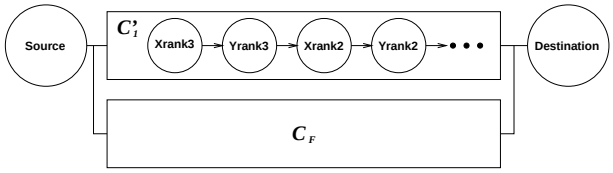

図 5: 3.a でのチャネル集合

3.b 3.a の方法では, 偶数ランク, 奇数ランクでの単位ベクトルは共通にしなければならなかったため, ベクトルルーティングによりあらかじめベクトルを求めたときに, ランク内で逆方向のベクトルを使用する必要が出たときにルーティングできなくなる可能性がある. そのため, 各ランクでの単位ベクトルを共通にしなくてもいい方法を考える.

バーチャルチャネル C_{Fn} を C'_1 全体ではなく, 各ランクの C_1 ごとに用意する (図 6). さらに, 使用するランクの順を C_1 と同様に制限する. これにより, 各ランク間での単位ベクトルを一致させる必要はなくなる.

それぞれのルーティング法でルーティングが可能なベクトル集合と可能でないものをそれぞれ図 7と図 8に示す.

図 7(a) のベクトルはあるルーティングの例である. これにベクトルの入れ換えを行ったものが (b) のベクトルである.3.a の方法では, ランクの使用順を自由に決められるため, このルーティングは可能である. しかし, 3.b の方法では, 図 8(c) にあるように, ルーティングすることは不可能である.

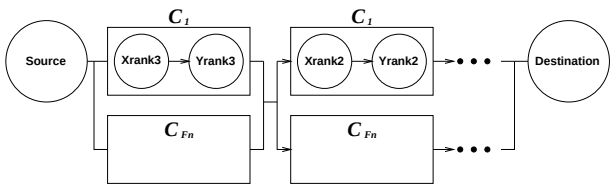

図 6: 3.b でのチャネル集合

図 7(c) のベクトルは, 奇数ランク (ランク 3, ランク 1) でベクトルが逆方向に向かっている例である. これは,3.a の方法ではルーティングすることは不可能である. 逆に,3.b の方法では, 図 8(b) に示すように, ルーティングすることは可能である.

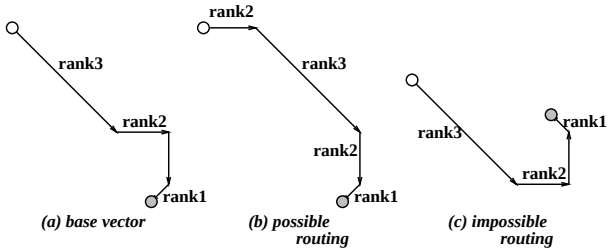

図 7: 3.a でのベクトル集合の例

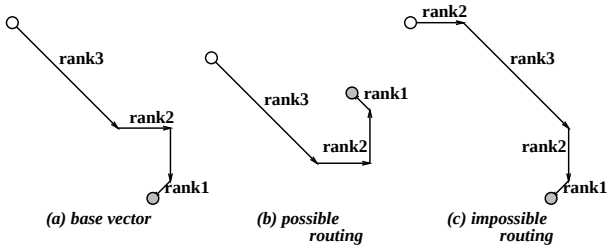

図 8: 3.b でのベクトル集合の例

5 Turn モデルを用いた方法

RDT(2,4,1)/αでは, あるノードが持っていないランクを利用するためには, 基本トーラスを移動しなければならない. ここで,3.a の方法では, 逆向きに方向転換ができないため, 結果的にランクを決められた順番に移動することになる. そこで,Turn モデルによる方法を提案する.

5.1 Turn モデル

デッドロックが生じるのは, 結合網内のバッファが論理的に循環構造を作ってしまうためである.Glass らによる Turn モデル [4] は, 循環を生じないように, パケットがルーティング中に方向を変えるパターンを制限する方法である.

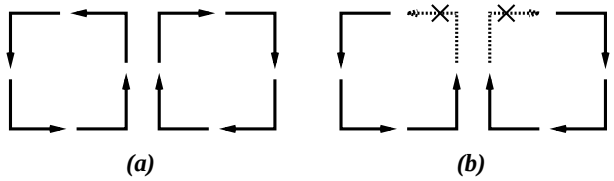

図 9: Turn モデル

ここでもっとも単純な 2 次元メッシュでの Turn モデルを考える. この場合, 可能な循環構造を, 図 9(a) に示す. ここで,Turn モデルに従い, 各循環の進路変更を 1 つずつ禁止し, デッドロックを防いだ場合を図 9(b) に示す. このルーティング方法は north-last 法と呼ばれる.

5.2 RDT での Turn モデル

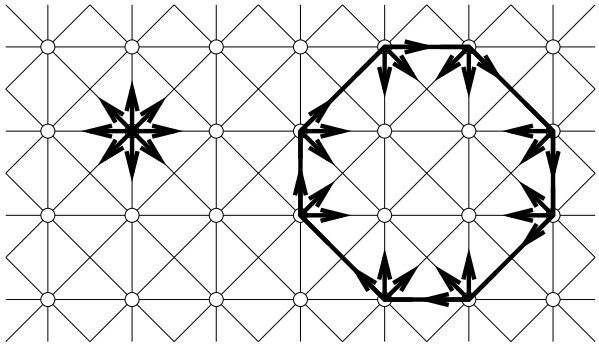

図 10: RDT での循環構造

先ほど述べた 2 次元メッシュでの Turn モデルを RDT でのモデルに拡張する. まず,RDT での可能な循環構造について考える.PRDT では図 10 で表されるように,8 方向に曲がることができるため, 図 10のような循環構造を示す.8 角形ではなく,3 角形,4 角形などの循環構造もとることが可能だが, 全てこの 8 角形の循環構造で示せるので省略する.

次に, 図 9と同様に, 循環を生じないようにルーティングの方向を制限する.north-last 法を用いるので, 循環構造の右上 (もしくは左上) となりうるベクトルの使用を

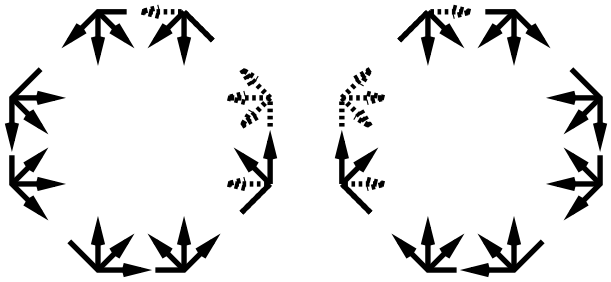

図 11: RDT での Turn モデル

禁止する. その結果, 図 11で示すようなベクトルの集合となる.

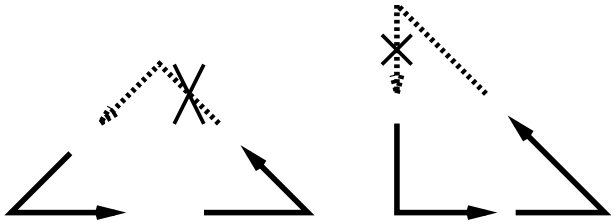

図 12: 特殊な場合の循環構造

以上の制限では不足している部分がある. すなわち, 循環構造は, 右上 (もしくは左上) の角がない場合にも生ずる. その例を図 12に示す. よって, 図 12で点線で示したベクトルの使用も禁止する. 以上により,RDT での循環構造は存在しなくなる. 新しく求めた RDT での Turn モデルを図 13に示す. このモデルは PRDT をもとに考えたが, RDT(2,4,1)/αでもモデルを変えることなく利用することが出来る.

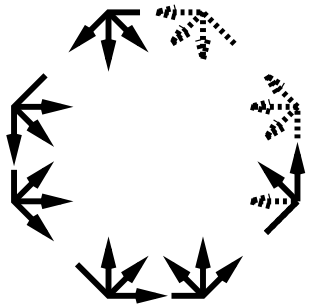

図 13: RDT での Turn モデル 2

一般的な 2 次元トーラスでの Turn モデルは, 図 9(b) を見てもわかるように, 曲がるパターンが 8 つあるうち,2 つを禁止する. つまり$\frac{1}{4}$の進路変更が使用不可とな

る. RDTでのTurnモデルでは,24パターン中7つを禁止するため, 2次元トーラスのときより若干効率が悪くなる. しかし,Duatoの必要十分条件を用いた方法では不可能だったRDT(2,4,1)/αでのランクの使用順を自由に決めれる点や, 同じ次元内で逆方向に進むことを可能にしており,adaptive routingの自由度としては高くなっている.

6 まとめ

adaptive routingを相互結合網RDTに2つのアプローチから適用し, 新しいルーティング法を提案した.

Duatoによる必要十分条件を用いて,PRDT用のadaptive routingを2種類提案した. これらの方法は, ネットワーク中の利用されていないチャネルを最大限に活用することが出来, 高い通過率を示すことが予想される. しかし,RDTでは使用するランクの順番を変えることが出来ず, また元来最短経路を通ることしか許していないため,adaptive routingとしての自由度は低い.

更に,Turnモデルを用いたadaptive routingを提案した. このルーティングはPRDT, RDT(2,4,1)/αのどちらにも使用することが出来る. また, ランクの使用順は自由, 最短経路を通る必要がない, 各次元内で逆方向に方向転換が可能という点でDuatoの方法よりも高い自由度を示す. しかし,north-last法を使用しているので, 北に進むのは最後にしなければならないといった制限はある.

提案したルーティング法はそれぞれ一長一短であり,RDT, ベクトルルーティングといった特殊な用途にどのルーティング法が向いているのかを決めるには, シミュレーションを動かして検討する必要がある. 既にdeterministic routingを用いたRDT(2,4,1)/αのルータチップは実装されており, 今後はadaptive routingを用いたチップの実装を想定したシミュレーションを行って評価及び検討を行う予定である.

参考文献

[1] W.J.Dally, C.L.Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. on Comput.*, vol. 36 no. 5, pp. 547-553, May. 1987.

[2] J. Duato , "A Necessary And Sufficient Condition For Deadlock-Free Adaptive Routing In Wormhole Networks" In *Proc. of the International Conference on Parallel Processing.*, vol.I, pp.142-149, 1994.

[3] J. Duato , "A Necessary And Sufficient Condition For Deadlock-Free Adaptive Routing In Wormhole Networks" In *IEEE Trans. on Parallel and Distributed Systems, Vol.6, No.10*, 1995.

[4] C.J.Glass and L.M.Ni, "Maximally Fully Adaptive Routing in 2D Meshes," In *Proc. of ISCA92.* pp.278-287, 1992.

[5] T.Tanaka, et, al. "The Massively Parallel Processing System JUMP-1," Ohmsha, 1996.

[6] P.Merlin, P.Schweitzer, Deadlock Avoidance in Store-and-Forward Networks-I : Store-and-Forward Deadlock, *IEEE Computer* Vol.28, No.3, pp.345-354, (1980).

[7] Y. Yang, H. Amano, H. Shibamura, and T.Sueyoshi. Recursive diagonal torus: An interconnection network for massively parallel computers. In *Proc. of IEEE SPDP*, 1993.

[8] Y. Yang, H. Amano, Message Transfer Algorithms on the RDT. In *IEICE Trans. on Information and Systems, Vol.E79-D, No.2*, 1996.

[9] T.Kudoh, et.al, Hierarchical bit-map directory schemes on RDT for a massively parallel processor JUMP-1 *Proc. of ICPP*, 1995.

[10] H.Nishi, K.Nishimura, K.Anjo, H.Amano, T.Kudoh, The JUMP-1 router chip *Proc. of International Phoenix conference on computers and communications*, 1996.

[11] 楊、天野、柴村、末吉、 超並列計算機向き結合網:RDT, 電子情報通信学会論文誌, *D-I Vol.J78-D-I*, 1995.

[12] 相互結合網RDT上で階層マルチキャストによるメモリコヒーレンシ維持手法, 情報処理学会論文誌, *37*巻 *7*号, 1996.