論　文

相互結合網RDTにおけるadaptive routing

舟橋　啓[†]　　上樂　明也[††]　　天野　英晴[††]

Adaptive routing for Recursive Diagonal Torus

Akira FUNAHASHI[†], Akiya JOURAKU[††], and Hideharu AMANO[††]

あらまし　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うために優れた性質を多く持っている．超並列計算機に用いられる相互結合網において，混雑もしくは故障したノードを回避できるルーティングアルゴリズムは必要不可欠である．既存のベクトルルーティングはシンプルではあるが，固定ルーティング（deterministic routing）であるためこの様なノードを迂回することはできない．
本研究では，Duatoによる必要十分条件を用いることによりRDT(2,4,1)/α上でデッドロックフリーなadaptive routing algorithm D-RDTとFD-RDTを提案し，デッドロックフリーであることを証明し，評価を行った．D-RDTは各ランク内のトーラスではfully adaptive routingを行うが，e-cube routingをベースにしているため使用するランク順が固定される．FD-RDTは自由にランクを使用することが可能なバーチャルチャネルを付加したことにより，D-RDTに比べルーティングの自由度は高い．
シミュレーションによる評価の結果，両アルゴリズムとも既存のdeterministic routingに比べ通過率，レイテンシ共に性能向上が見られ，特にFD-RDTを用いたルーティングでは顕著であった．
キーワード　相互結合網，適応型ルーティング，デッドロックフリー，RDT

1. はじめに

RDT(Recursive Diagonal Torus) [2]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，既にルータチップが開発され[4]，超並列計算機テストベッドJUMP-1 [3]上で稼働している[1]．

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が提案され[5], [6]，JUMP-1上で実装されている．

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[2]，e-cubeルーティング[7]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まっているため，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，評価する．

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[9]を参照されたい．

RDTは二次元トーラスを基本とし，対角線上に再帰的に構成した異なる大きさのトーラス状結合網の組合せにより構成される．

一般に，トーラス構造に対してバイパスリンクを定

[†]三重大学工学部，津市
Dept. of Information Engineering, Mie Univ., 1515, Kamihama, Tsu, 514–8507 Japan
[††]慶應義塾大学理工学部，横浜市
Dept. of Computer Science, Keio Univ., Kanagawa, 223–8522 Japan

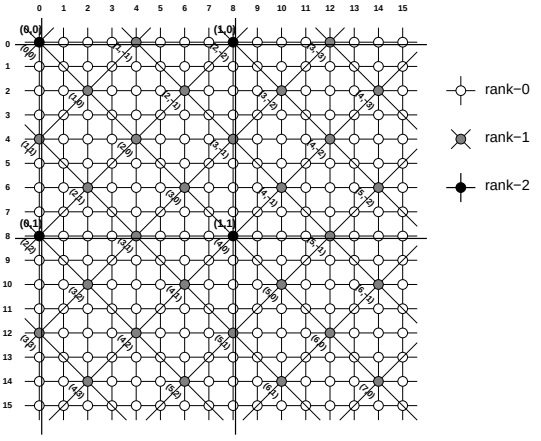

図 1　上位ランクの構成例
Fig. 1　Structure of the Upper rank torus

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．

この新たに形成された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．これに習い，基本トーラスはランク 0 トーラスと呼ぶ．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．基数 n を 2 にした場合のランク 1 トーラスとランク 2 トーラスの例を図 1に示す．図 1に示すように，各ランクのトーラスにはそれぞれ x 軸，y 軸の座標軸を持つ．以降 x 方向，y 方向と記されている場合は絶対座標ではなく，各ランクのトーラスにおける方向であることを注意されたい．

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．

ランク 0 トーラス（基本トーラス）に対し再帰的に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 1］：完全 **RDT**
全体のノード構成が $k \times k$ のランク 0 トーラス（基本トーラス）上に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．最大ランク数 R は k の値によって変化し，以下の式を満たすような値をとる．

$$n^{\left\{\frac{3R}{2}+z\right\}} \leqq k < n^{\left\{\frac{3}{2}(R+1)+z\right\}}$$

$$z = \begin{cases} 0, & R \text{ が偶数} \\ -\frac{1}{2}, & R \text{ が奇数} \end{cases}$$

□

図 1ではランク 0 の（0，0），(4，0），(0，4），(4，4）等のノードにしかランク 1 トーラスが構成されていないが，完全 RDT では（0，1），(1，0），(1，1）等のノードにも同様にランク 1 トーラス，ランク 2 トーラス等の上位トーラスが構成される．

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 2］：**RDT**
ランク 0 トーラス（基本トーラス）上に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，$n = 2$，$R = 4$（degree が 8），$m = 1$ に取るのが有利である．RDT では基本トーラスでの隣り合

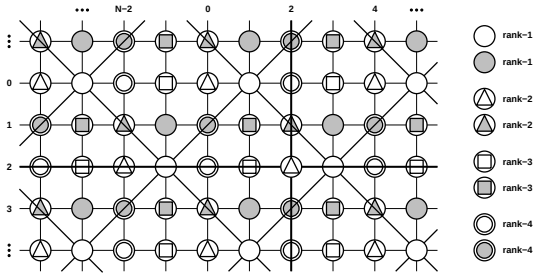

図 2　RDT(2,4,1)/α の構成
Fig. 2　Torus assignment for the RDT(2,4,1)/α

う各ノードは異なる上位ランクトーラスを選択可能なため，RDT(2，4，1) は様々な構成を持つことが可能である．上位ランクトーラスの配置は様々なパターンが考えられるが，有利であると考えられる構成は RDT(2,4,1)/α と呼ばれるものである．JUMP-1 では図 2に示す構成の RDT， RDT(2,4,1)/α が採用されている．

$n = 2$ の場合， あるランク上には独立なトーラスが 8 つ形成される． RDT(2,4,1)/α も $n = 2$ であるから，基本トーラス上に 8 つのランク 1 トーラスが形成される．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，それぞれのランクに対してトーラスを 2 つずつ用いる． 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．

この構成では図 2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．

2.1　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．

ここでは紙面の都合により直観的なモデルのみを示すが，詳細な定義については [9] を参照されたい．

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，$\vec{A} = a\vec{x_0} + b\vec{y_0}$ として表される．ここで a と b は，ノード番号の差により， 簡単に求められる．まず， 各上位トーラスの単位ベクトルの方向を 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．

$$\vec{x}_{i+1} = n\vec{x}_i + n\vec{y}_i$$

$$\vec{y}_{i+1} = -n\vec{x}_i + n\vec{y}_i$$

目的とするベクトル $a\vec{x_0} + b\vec{y_0}$ を， 上位トーラスに相当するベクトルと， ランク 0 トーラスのベクトルの和の形で表す．

$$a\vec{x_0} + b\vec{y_0} = g\vec{x_1} + f\vec{y_1} + j\vec{x_0} + k\vec{y_0}$$

ここで， 上位トーラスに相当するベクトル g と f は以下のように定められる．

$$g = \frac{a+b}{2n},\ f = -\frac{a-b}{2n}$$

ここで， j と k は， ランク 1 トーラスの単位ベクトルより小さい範囲で，ランク 0 トーラスを用いる必要があるルーティングを示す．ランク 0 トーラスにおけるベクトルの絶対値を小さくするため，ここで行なう整数除算は， 切捨てではなく 2 捨 3 入を用いる．

$$\begin{aligned} a\vec{x_0} + b\vec{y_0} \ =\ & g(n\vec{x_0} + n\vec{y_0}) + \\ & f(-n\vec{x_0} + n\vec{y_0}) + j\vec{x_0} + k\vec{y_0} \end{aligned}$$

$$a = ng - nf + j, \quad b = ng + nf + k$$

$$j = a - ng + nf, \quad k = b - ng - nf$$

ここで， g と f 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．

例として，$n = 2$ とした場合の（1，2）から（5，9）へのルーティングのベクトル分解を図 3に示す．

図 3の Step 1 では（1，2）から（5，9）までの $\vec{A}$ をランク 0 の単位ベクトルであらわしている． 順に Step 2， Step 3 とベクトル分解を繰り返すと，Step 1 では 11 hop かかっていたルーティングが 4 hop で可能となる．

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 [10] 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

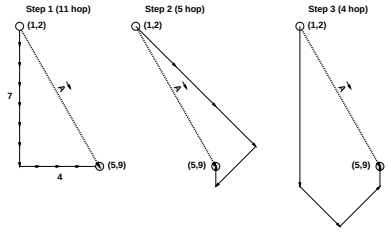

図 3　ベクトル ルーティングの例
Fig. 3　An example of the vector conversion

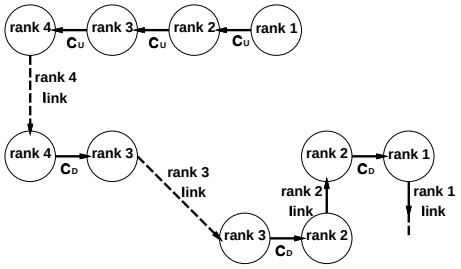

図 4　RDT(2,4,1)/α での e-cube routing
Fig. 4　The e-cube routing on the RDT(2,4,1)/α

ングは決定している．そのため一度経路が決定した後は経路中の混雑，故障箇所を迂回することができない．

2.2　RDT(2,4,1)/α におけるデッドロックフリールーティング

e-cube routing [7] は k-ary n-cube での一般的なデッドロックフリールーティングアルゴリズムとして知られている．RDT は k-ary 2-cube を再帰的に結合させたネットワークなので，e-cube routing を適応させることができる．ここで e-cube routing について定義する．

[定義 3] ：**e-cube routing**

e-cube routing では，出発地ノードから目的地ノードまでのルーティングを次元順に行う．各次元 i ではパケットは目的地ノードの i 次元の座標が等しくなるまでルーティングされる．次元順にルーティングを行うため，パス上で使用されるチャネルには循環構造が発生せず，結果デッドロックフリーを保証する [7]．□

e-cube routing は完全 RDT には簡単に適応させることができる．各ランクでの k-ary 2-cube で e-cube routing を用いることにより，各ランク内ではデッドロックフリーが保証される．完全 RDT は各ノードがすべての上位トーラスへのリンクを持っているため，各ランク間の移動の順番を固定する（ランクが高い k-ary 2-cube から順に使用する）ことにより，完全 RDT におけるベクトルルーティングはデッドロックフリーであることが保証される．

RDT(2,4,1)/α では上記の手段ではデッドロックフリーを保証することができない．なぜなら RDT(2,4,1)/α では各ノードがただ一つの上位トーラスへのリンクを持っているだけなので，使用するベクトルのランクを変更するためには基本トーラス（ランク 0 トーラス）を用いてランク間の移動を行なわなければならないからである．このランク間の移動をローカルルーティングと呼ぶ．完全 RDT でのベクトルルーティングにはバーチャルチャネルは必要ないが，RDT(2,4,1)/α ではローカルルーティングを行なう時にもデッドロックフリーを保証するために，2 つのバーチャルチャネルを付加し，ベクトルルーティングアルゴリズムの拡張を行なう．

[定義 4] ：**RDT(2,4,1)/α におけるベクトルルーティング**

（1） 図 2に示すように， RDT(2,4,1)/α では基本トーラスの x 次元の正方向に向かって上位ランクから下位ランクを持つノードが並ぶ．

（2） 2 つのバーチャルチャネル $C_U(up)$ と $C_D(down)$ を付加する．C_U は $-x$ 方向のチャネルに，C_D は $+x$ 方向のチャネルとしてそれぞれ基本トーラスに付加する．

（3） 定義 3に示した e-cube routing を用い，ベクトルをランクが高いものから順に使用する．もしパケットがルーティングを開始するべき最高位ランク（R）になければ，まずチャネル C_U を用いて $-x$ 方向に進み，最高位ランクへ送られる．

（4） パケットが R でのルーティングを終えたならば C_D を用いて $+x$ 方向に進み，パケットはランク $R-1$ のトーラスでのルーティングを行なう．以下，最下位ランクのトーラスでのルーティングが終了するまで繰り返される．

このアルゴリズムがデッドロックフリーであることは [8] にて証明されている．図 4にこのアルゴリズムの例を示す．

3.　RDT(2,4,1)/α 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る．RDT で用いられているベクトルルーティングはベクトルの順序を入れ替えるだけで代替経路が得られるため，adaptive routing を適用するには有利である．

adaptive routing も deterministic routing 同様，デッドロックを起こす可能性がある [10]．今まで様々なデッドロックフリー adaptive routing が提案されてきたが，大別すると最短経路を使用するものと冗長なステップをも許すものに分けられる．そこで，ここでは 2 つのアプローチから RDT(2,4,1)/α 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，最短経路のみを使用する場合とそうでない場合の 2 種類のルーティングアルゴリズムに大別される．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ため，自由度は高くなる．

以下，最短経路のみを使用する Duato's protocol による方法を説明し，RDT(2,4,1)/α における adaptive routing を提案する．

3.1 k-ary n-cube における Duato's protocol

今まで提案されてきたデッドロックフリーな adaptive routing は，何らかの方法で経路の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 [11]．

Duato の条件は，

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

（2） escape path により循環が切断されてデッドロックが発生しない経路と，escape path の二種類の経路を用いて結合網中のどのノード間でもパケットが送れることを保証する

の 2 点を満足できれば (1),(2) の経路を adaptive に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．Duato はこの方法を k-ary n-cube に適用し，デッドロックフリーであることを証明した．[12]．以下，簡単に手順を説明する．

（1） はじめに単方向リング（図 5）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 と C_A を使用し，低い場合は C_A のみを使用することで 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．定義 3に示した e-cube routing 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 $\mathbf{C_1}$ 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．

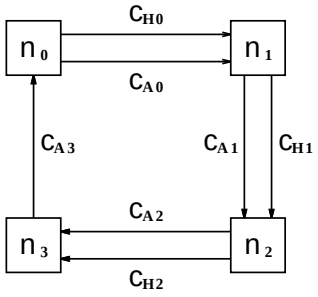

図 5　Duato's protocol のリング状の結合網での適用例
Fig. 5　An example of Duato's protocol for ring network

ここで k-ary n-cube 上での Duato's protocol を定義する．

［定義 5］ **k-ary n-cube 上での Duato's protocol**

k-ary n-cube 上での Duato's protocol では以下のチャネル，パス，ルーティングアルゴリズムを用いる．

- channel
  - C_H : 各次元において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にのみ利

用可能なチャネル．定義3に示した e-cube routing のみ利用可能

– C_A ：各次元内での目的地ノード番号の大小によらず利用可能．定義3に示した e-cube routing のみ利用可能なチャネル

– C_F ：各次元内での目的地ノード番号の大小によらず利用可能なチャネル．次元順を無視したルーティング時にも利用可能な fully adaptive チャネル．

• path

– $\mathbf{C_1}$ ：定義3に示した e-cube routing を用いて次元順に各次元のチャネルを使用したパス．各次元内では目的地ノード番号により C_H，C_A を使い分ける．$\mathbf{C_1}$ にはチャネル C_F は含まれない．$\mathbf{C_1}$ はデッドロックフリーであり，これを escape path と呼ぶ．

• routing : k-ary n-cube 中の全てのノードが全ての目的地ノードに対してデッドロックフリーなパス $\mathbf{C_1}$ を持つため，このルーティングアルゴリズムはデッドロックフリーである．そこで新たなチャネル C_F を次元順を無視したルーティングに用いる．パケットがルーティングされる時に，動的に C_H，C_A，C_F のいずれかを各チャネルの制限事項に沿って選択し，利用する．チャネル C_F を用いる場合には最短経路上ならばどの次元方向にも進んでもよい． □

3.2 RDT(2,4,1)/α における Duato's protocol

ここでは k-ary n-cube 上での adaptive routing を拡張し，RDT(2,4,1)/α に適用し，アルゴリズム D-RDT と FD-RDT を提案する．D-RDT，FD-RDT は RDT(2,4,1)/α に限らず，図2に示したようなトーラス構成を持つ RDT(n,R,m)/α に対し利用可能であるが，本論文では混乱を避けるため，既に実装され動作実績もある RDT(2,4,1)/α での adaptive routing について述べる．

3.2.1 D-RDT （Duato's protocol on RDT）

［定義 6］ **Algorithm: D-RDT**

D-RDT では以下のチャネル，パス，ルーティングアルゴリズムを用いる．

• channel

– C_H，C_A ： 定義 5 同様，各ランク内でのトーラスにおいて e-cube routing にのみ用いられるチャネル．

– C_U，C_D ： 定義 4 同様，ランク間の移動（$\pm x$ 方向）に用いられるチャネル．

– C_{Fn} ： 定義 5の C_F と同様，各ランク内でのトーラスにおいて利用されるチャネル．C_{Fn} は定義 5 と同様，各ランク内のチャネル集合 $\mathbf{C_1}$ に対して用意される fully adaptive チャネルである（図 6）．

• path

– $\mathbf{C_1}$ ： 定義 5と同様，各ランク内でのトーラスにおいて定義3に示した e-cube routing を用いたチャネル集合．チャネル C_H，C_A のみを用いているため，各ランク内ではデッドロックフリーな escape path となるが，あくまで各ランク内のトーラスでのデッドロックフリーのみを保証する．

– $\mathbf{C'_1}$ ： RDT(2,4,1)/α でのルーティングを行う際に，ランクの使用順を固定したときに得られるチャネル集合．C_{xi}，C_{yi} をランク i での $\pm x$，$\pm y$ 次元方向のチャネルとすると，チャネルを

$C_{x3} \rightarrow C_{y3} \rightarrow C_{x2} \rightarrow C_{y2} \rightarrow C_{x1} \rightarrow C_{y1}$

（ランク 3 の $\pm x$ 方向から順に $\pm y$ 方向，ランク 2 の $\pm x$ 方向$\cdots$）という順に使用するように制限する．また，ランク間の移動にはランク間の移動用のチャネル C_U，C_D を使用する．ルーティングを始めるにあたって出発地ノードが最上位ランクのトーラスを持たない場合は C_U を使用して最上位ランクを持つノードへ移動してからルーティングを開始する．つまり，実際には $\mathbf{C'_1}$ は

$C_U \rightarrow C_U \rightarrow C_{x3} \rightarrow C_{y3} \rightarrow C_D \rightarrow C_{x2} \rightarrow C_{y2} \cdots$

のようなチャネル集合となる．ランクの使用順を固定しており，また各ランク内のチャネル集合 $\mathbf{C_1}$ は各ランク内での escape path となるため，$\mathbf{C'_1}$ は RDT(2,4,1)/α での escape path となる．

• routing : ルーティングを行う際，チャネル集合 $\mathbf{C'_1}$ に沿ってチャネル C_U を利用して最上位ランクにまで移動しルーティングを開始する．ランク内でのルーティングが終了したならばランク順に従いチャネル C_D を用いて次のランクに移動し，順次各ランクでのルーティングを行う．各ランク内ではチャネル集合 $\mathbf{C_1}$ とチャネル C_{Fn} があるため，定義 5 同様に動的にチャネル C_{Fn},C_H,C_A を選択してルーティングを行う．チャネル C_{Fn} を用いる場合には最短経路上ならばどの次元方向にも進んでもよい． □

C_{Fn} も $\mathbf{C_1}$ 同様，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．また，使用するランク順は固定となる．このアルゴリズム D-RDT 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図 7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．

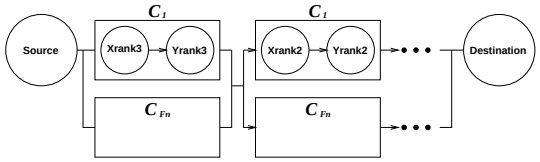

図 6　fully adaptive チャネル C_{Fn} の配置法
Fig. 6　An arrangement of fully adaptive virtual channel C_{Fn}

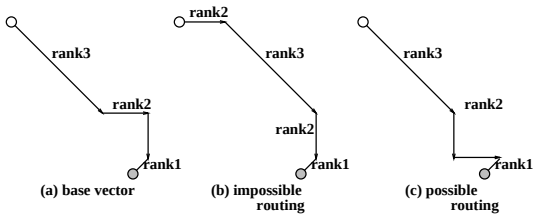

図 7　D-RDT でのルーティング例
Fig. 7　Examples of vectors in algorithm D-RDT

これにベクトルの入れ換えを行ったものが（b），(c) のベクトルである．D-RDT はランクの使用順を固定としているため，(b）のベクトルのようにランク 2 のベクトルを使用する前にランク 3 のベクトルを使用するようなルーティングは不可能である．一方，(c）のベクトルのような，ランクの使用順を守ったルーティングは D-RDT では可能となる．

［定理 1］ アルゴリズム D-RDT はデッドロックフリーである．

［ 証明 ］ チャネル集合 $\mathbf{C_1}$ は定義 5 で用いられている escape path と同一のものであり，デッドロックフリーである．また，C_{Fn} は各トーラス内での最短経路である．定義 5より，各ランクのトーラスではアルゴリズム D-RDT はデッドロックフリーである．ランクの使用順は固定されており，その使用順は定義 3に示される e-cube routing のそれと等しいため，全てのランク内で $\mathbf{C_1}$，C_{Fn} は循環構造を作らない．よって，アルゴリズム D-RDT はデッドロックフリーである．　□

D-RDT ではランクの使用順が固定されているため，RDT(2,4,1)/α では問題が生じる．なぜならルーティングに用いられるベクトルは最上位ランクのものから順に下位のランクを使用する制限があるが，RDT(2,4,1)/α は各ノードは一つの上位ランクトーラスしか持たないからである．そのため使用したいランクのトーラスがそのノードに存在しない場合は 2 章で述べたように C_U，C_D を用いてランク間の移動（ローカルルーティング）を行なわなければならない．

k-ary n-cube では Duato's protocol は最低でも C_H，C_A の 2 つのバーチャルチャネルを必要とする．D-RDT では更に C_{Fn} を必要とし，ローカルルーティングのために $-x$ 方向に C_U もしくは $+x$ 方向に C_D を必要とする．つまり，D-RDT ではバーチャルチャネルが $\pm x$ 方向に C_H，C_A，C_U（もしくは C_D），C_{Fn} の 4 つ，$\pm y$ 方向に C_H，C_A，C_{Fn} の 3 つが必要である．

3.2.2　FD-RDT（Fully adaptive D-RDT）

Duato's protocol を完全 RDT において使用する場合には比較的容易に利用することができたが，RDT(2,4,1)/α はその複雑な構成により結果的に D-RDT にはベクトルの使用順等の制限が多く加わり，fully adaptive routing [13] を実現することは不可能であった．そこで，RDT(2,4,1)/α を含む RDT(n,R,m)/α に適用可能な fully adaptive routing アルゴリズムである FD-RDT を提案する．

［定義 7］ **Algorithm: FD-RDT**

FD-RDT では以下のチャネル，パス，ルーティングアルゴリズムを用いる．D-RDT との相違点はチャネル C_R の有無，及びその使用方法である．チャネル C_H，C_A，C_U，C_D，C_{Fn}，チャネル集合 $\mathbf{C_1}$，$\mathbf{C_1'}$ は定義 6にて定義されているものと全く等しい．

- channel
  - C_H，C_A：定義 5，定義 6 同様，各ランク内でのトーラスにおいて e-cube routing にのみ用いられるチャネル．
  - C_U，C_D：定義 4，定義 6 同様，ランク間の移動（$\pm x$ 方向）に用いられるチャネル．
  - C_{Fn}：定義 5の C_F 及び，定義 6 の C_{Fn} と同様，各ランク内でのトーラスにおいて利用されるチャネル．
  - C_R：基本トーラス（ランク 0 トーラス）上に設けられたランク間の移動のためのチャネル．C_U，C_D と異なり基本トーラス上の全方向（$\pm x, \pm y$ 方向）に用意される．ランク間の移動に C_R を用いる場合，ランクの使用順を無視した移動が可能．更に，$\pm x$ 方向のみによるランク間の移動だけでなく，$\pm y$ 方向によるランク間の移動も可能．
- path
  - $\mathbf{C_1}$：定義 6と同様のチャネル集合．各ランク内ではデッドロックフリーな escape path となるが，あ

くまで各ランク内のトーラスでのデッドロックフリーのみを保証する．

– $\mathbf{C}'_1$：定義6と同様のチャネル集合．ランクの使用順を固定しており，また各ランク内のチャネル集合 $\mathbf{C_1}$ は各ランク内での escape path となるため，$\mathbf{C}'_1$ は RDT(2,4,1)/α での escape path となる．

• routing：FD-RDT でのルーティングでは，チャネル C_R が利用可能な状態である限り，ランクの使用順を守る必要は無い．C_R はランクの使用順を守らずにランク間の移動（ローカルルーティング）を行うことを許すだけでなく，(C_U，C_D を用いたローカルルーティングは $\pm x$ 方向のみである一方）C_R は $\pm x$，$\pm y$ 両方向への移動によるローカルルーティングを許す．各ランク内のトーラスでのルーティングは定義6の各ランク内のトーラスでのルーティングと同様，チャネル C_H，C_A，C_{Fn}，を動的に選択することにより行う．C_R を利用する限りランクの使用順を守る必要はないが，デッドロックを防ぐために「一度 C_U を用いてローカルルーティングを行なったら，それ以降は C_R を用いて下位のランクに移動してはならない」という制限を加える（C_U を利用した後に C_R を利用して上位のランクに移動するのは構わない）． □

C_R は C_U，C_D と異なり，ランク間の移動順を守らずにランク間の移動（ローカルルーティング）を行うことを許し，かつ $\pm x$，$\pm y$ 両方向への移動によるローカルルーティングを許すため，C_R と C_{Fn} を用いることにより，FD-RDT では fully adaptive routing を実現する．

C_R の使用を制限する理由は，C_U と C_D の使用順を固定するためと，C_U を利用する回数を制限するためである．$\mathbf{C}'_1$ に含まれる C_U，C_D は使用回数が固定されており，あるランク i からランク $i+1$ に用いられる C_U はただ一つのみであることによりデッドロックフリーを保証している．C_U を利用した後に C_R を利用して低いランクに移動することは結果的に再びランク i からランク $i+1$ への移動に C_U を利用する可能性を残すため，デッドロックを引き起こす可能性がある．

[定理 2] アルゴリズム FD-RDT はデッドロックフリーである．

[証明] $\mathbf{C}'_1$ は定義6にて定義されているものと等しいため，定理1よりデッドロックフリーである．また，D-RDT はデッドロックフリーである．D-RDT は利用するランク順を固定することによりデッドロックフリーを保証している．つまり，C_U，C_D の使用回数が固定であり，なおかつ C_D は C_U を使いきった後のみ使用可能という制限がある．C_R はランク順を無視して使用することが可能であるが，この C_U，C_D の使用回数及び使用順の制限を守るよう，C_R の利用をある程度禁止しなければならない．そこで C_R に「一度 C_U を用いてローカルルーティングを行なったら，それ以降は C_R を用いて下位のランクに移動してはならない」という制限を加える．これによりチャネル集合 $\mathbf{C}'_1$ に含まれる C_U，C_D の使用順に影響を及ぼさず，$\mathbf{C}'_1$ 中の C_U，C_D の使用回数を増やすこともない．よって FD-RDT でも定理 1で証明されたデッドロックフリーなパス（escape path）が全ての目的地ノードに対し少なくとも一つ存在する．よって，アルゴリズム FD-RDT はデッドロックフリーである． □

D-RDT と異なり，FD-RDT は図7中の全てのルーティングが可能であり，D-RDT と比べより多くの代替経路を選択することが可能である．

FD-RDT では D-RDT に比べ新たにバーチャルチャネル C_R を必要とするため，バーチャルチャネルは $\pm x$ 方向に C_H，C_A，C_U（もしくは C_D），C_{Fn}，C_R の5つ，$\pm y$ 方向に C_H，C_A，C_{Fn}，C_R の4つが必要である．

4. 評　　価

ここでは，提案された2つの adaptive routing についてシミュレーションを実行し，deterministic routing である e-cube routing との比較を行った．以下，シミュレーションに用いた条件を示す．

4.1 RDT シミュレータ

シミュレータは C++言語で記述され，フリットレベルのシミュレーションを行う．本シミュレータはパケット転送方式に wormhole を用い，ネットワークサイズ，バーチャルチャネル数，パケット長はパラメータを変更することにより設定する．図8に示すように，各ノードはプロセッサ，リクエストキュー，ルータにより構成されている．各ノードはルータに接続された双方向チャネルにより隣接ノードと接続されている．

4.2 ルータモデル

図9に示されるように，ルータはチャネルバッファ，クロスバ，リンクコントローラ（LC），バーチャルチャネルコントローラ（VC），アービタ，制御回路から構成されている．

channel selection policy には input selection pol-

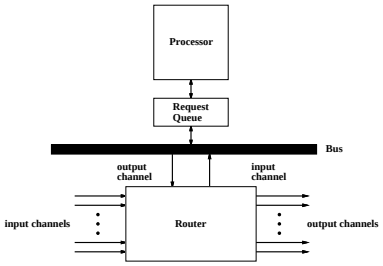

図 8　各ノードの構成
Fig. 8　The construction of each node

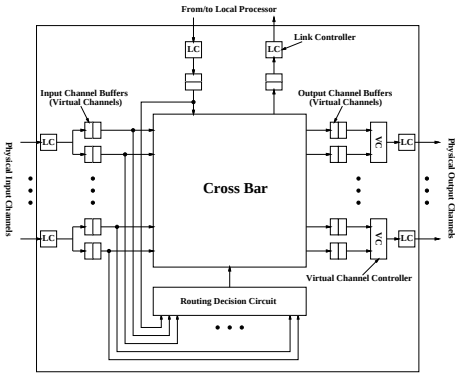

図 9　ルータモデル
Fig. 9　Router model

icy，physical channel transmission policy をそれぞれ設定することが可能である．

4.3　シミュレーション条件

本シミュレータでは目的地ノードは以下のトラフィックパターンにより決定する．

- uniform

全ての目的地ノードはランダムに決定され，均一に分散される．

- bit-reversal

ノード (x, y) は各自のノード番号 $k \times y + x$ をビット列に変換し，そのビット列を逆順に並べたノード番号を持つノードへパケットを送る．例えば 4×4 のネットワークサイズならば（$k = 4$），ノード $(2, 0)$ はノード番号が $4 \times 0 + 2 = 2$ となるのでビット列は [0010] となり，ビット列を逆順に並べた $[0100] = 4$ のノード $(0, 1)$ にパケットを送る．

- matrix transpose

ノード (x, y) は $(k - y - 1, k - x - 1)$ のノードにパケットを送る．

以下の二つの指標により評価を行なった．

4.3.1　ネットワークレイテンシ

あるノード p がパケットの最初のフリットを入力バッファに挿入した時刻を t_0，目的地ノードである q がパケットの最後のフリットを受け取った時刻を t_1 とする．ここで，$T_{lat}(p, q) = t_1 - t_0$ をネットワークレイテンシと呼び，ネットワークの性能を測る指標とする．

4.3.2　ネットワーク負荷

ここでは，ネットワークの負荷 L_n をパケット中のフリットがノードからルータに送られる確率と定義する．すなわち，フリットが毎クロック，ルータに送られるならば，ネットワークの負荷は 1.00 である．リクエストキューがあふれた場合は，シミュレーションは停止する．

表 1にシミュレーションのパラメータを示す．なお，128×128 ノードの RDT(2,4,1)/α のシミュレーションには 1Gbyte 以上の莫大なメモリを必要とするため，ネットワークサイズは 32×32 の RDT(2,3,1)/α，最大ランクは 3 で行なった．

4.4　シミュレーション結果

トラフィックパターンを uniform traffic，bit-reversal，matrix transpose とした場合の実行結果をそれぞれ図 10，図 11，図 12に示す．これらの図は，横軸にネットワーク負荷，縦軸にネットワークレイテンシを示している．

表 1　シミュレーション条件
Table 1　Simulation parameters

ネットワークサイズ	32 × 32（RDT(2,3,1)/α）
パケット長	16 flits（固定）
パケットヘッダ長	2 flits（固定）
パケット転送方式	wormhole
リクエストキューサイズ	100 packets
パケット生成時間	1 clock
ルーティング，クロスバ設定時間	1 clock
フリット転送時間 （input 〜 output buffer 間）	1 clock
フリット転送時間 （物理リンク間）	1 clock
シミュレーション時間	10000 clock （初めの 1000 clock は無視）
Input selection policy	distance traveled
Physical channel transmission policy	round robin
バーチャルチャネル数	x:3, y:2（e-cube） x:4, y:3（D-RDT） x:5, y:4（FD-RDT）

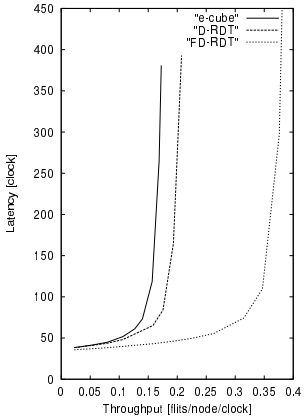

図 10　シミュレーション結果: uniform traffic
Fig. 10　Simulation result: uniform traffic

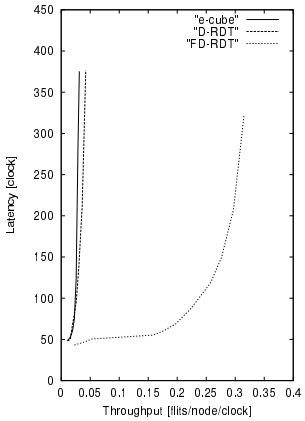

図 11　シミュレーション結果: bit-reversal
Fig. 11　Simulation result: bit-reversal

図 10より，uniform traffic では 2 つの adaptive routing，D-RDT，FD-RDT の方が deterministic（e-cube） routing よりも高い性能を示すことがわかる．更に，FD-RDT の方が D-RDT よりも高い性能を示していることがわかる．これは FD-RDT のより高い自由度が効果をあげていると言える．

図 11に bit-reversal traffic でのシミュレーション結果を示す．D-RDT と e-cube ではさほど性能差はないが，FD-RDT は高い性能を示している．bit-reversal では各ノードは毎回同じ目的地ノードに向けてパケットを送るため，ブロックされたパケットは他のパケットの進行を常に妨害する．そのため，代替経路を使うことが出来ない e-cube routing ではパケットのブロックは頻繁に発生する．D-RDT は adaptive routing ではあるが，ランク間の移動順が固定されているため各ランク内での 2 本程度の代替経路しか得ることが出来ない．逆に D-RDT では多くの代替経路を使用することが出来るため，この様な原因による性能低下は見られない．

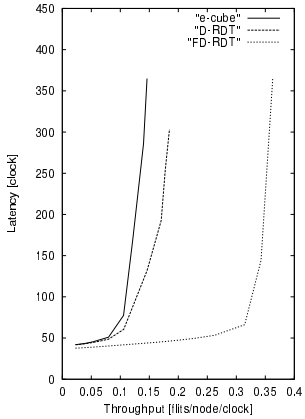

図 12　シミュレーション結果: matrix transpose
Fig. 12　Simulation result: matrix transpose

更に図 12に示すように，matrix transpose でも 2 つの adaptive routing は deterministic routing に比べて高い性能を示している．FD-RDT は他の 2 つのルーティングアルゴリズムに比べより高い性能向上が見られる．matrix transpose traffic ではノード (x, y) はノード $(k-y-1, k-x-1)$（k は各次元のノード数）にパケットを転送する．この場合，2 つの近隣のノードが激しく交信を行ない，近隣のノード間では e-cube や D-RDT には代替パスが少ないためパフォーマンスが悪い．FD-RDT ではこのような近隣ノード間の交信には C_R が使用され，結果的に高い性能を示す．

D-RDT は 3，もしくは 4 つ，FD-RDT は 4，もしくは 5 つのバーチャルチャネルを各物理リンク毎に必要とする．FD-RDT の方がバーチャルチャネルを一つ多く使用するが，圧倒的なパフォーマンスの向上を考慮すると，RDT(2,4,1)/α には FD-RDT を用いた adaptive routing が有効であると言える．

5. ま と め

RDT(2,4,1)/α における 2 つの adaptive routing，D-RDT，FD-RDT を提案，デッドロックフリーであることを証明した．

提案したルーティングアルゴリズムについてシミュレーションによる評価を行った．シミュレーション結果より，adaptive routing は 3 つの traffic pattern においてネットワークレイテンシ，スループット共に性能向上を示した．ハードウェア量を考慮すると RDT(2,4,1)/α 上での adaptive routing には，FD-RDT が最も適していると考えられる．

謝辞　本研究の一部は日本学術振興会特別研究員科学技術研究費補助金によるものである．

文　献

[1] 秤谷 雅史，小西 将人，田中 利彦，鈴木 紀章，佐藤 永子，井上 浩明，五島 正裕，天野 英晴，中島 浩，平木 敬，富田 眞治，“超並列計算機プロトタイプ JUMP-1 の性能評価”，並列処理シンポジウム JSPP'99 論文集，pp.225–225，Jun 1999.

[2] Y. Yang, H. Amano, H. Shibamura, T. Sueyoshi, “Recursive diagonal torus: An interconnection network for massively parallel computers”, Proc. of IEEE SPDP, pp.591–594, Dec 1993.

[3] T. Tanaka, et.al, “The Massively Parallel Processing System JUMP-1”, IOS Press, 1996.

[4] H. Nishi, K. Nishimura, K. Anjo, H. Amano, T. Kudoh, “The JUMP-1 router chip: The versatile router for supporting distributed shared memory”, Proc. of International Phoenix conference on computers and communications, pp.158–164, Mar 1996.

[5] T. Kudoh, H. Amano, T. Matsumoto, K. Hiraki, “Hierarchical bit-map directory schemes on RDT for a massively parallel processor JUMP-1”, Proc. of International Conference on Parallel Processing, Vol.1, pp.186–193, Aug 1995.

[6] 西村 克信，工藤 知弘，西 宏章，楊 愚魯，天野 英晴，“相互結合網 RDT 上での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，Vol.37, No.7, pp.1367–1377, Jul 1996.

[7] W. J. Dally, C. L. Seitz, “Deadlock-Free Message Routing in Multiprocessor Interconnection Networks”, IEEE Trans. on Computers, Vol.36, No.5, pp.547–553, May 1987.

[8] Y. Yang, H. Amano, “Message Transfer Algorithms on the RDT”, IEICE Trans. on Information and Systems, Vol.79, No.2, pp.107–116, Feb 1996.

[9] 楊 愚魯，天野 英晴，柴村 英智，末吉 敏則，“超並列計算機向き結合網 RDT”，電子情報通信学会論文誌，Vol.J78-D-I, No.2, pp.118–128, Feb 1995.

[10] L. M. Ni, P. K. McKinley, “A Survey of Wormhole Routing Techniques in Direct Networks”, IEEE Trans. on Computers, Vol.26, pp.62–76, Feb 1993.

[11] J. Duato, “A Necessary And Sufficient Condition For Deadlock-Free Adaptive Routing In Wormhole Networks”, Proc. of the International Conference on Parallel Processing, Vol.1, pp.142–149, Aug 1994.

[12] J. Duato, “A Necessary And Sufficient Condition For Deadlock-Free Adaptive Routing In Wormhole Networks”, IEEE Trans. on Parallel and Distributed Systems, Vol.6, No.10, pp.1055–1067, Oct 1995.

[13] J. Duato, S. Yalamanchili, L. Ni, “Interconnection Networks an Engineering Approach”, IEEE Computer Society Press, 1997.

（平成年月日受付，月日再受付）

舟橋　啓

平成 7 年慶應義塾大学理工学部電気工学科卒業．平成 12 年同大大学院理工学研究科計算機科学専攻博士課程修了．現在三重大学工学部情報工学科助手．工学博士．並列計算機の相互結合網の研究に従事．

上樂　明也

平成 10 年慶應義塾大学理工学部電気工学科卒業．平成 12 年同大大学院理工学研究科計算機科学専攻修士課程修了．並列計算機の相互結合網の研究に従事．

天野　英晴　（正員）

昭和 56 年慶應義塾大学工学部電気工学科卒業．昭和 61 年同大大学院工学研究科電気工学専攻博士課程修了．現在慶應義塾大学理工学部情報工学科助教授．工学博士．計算機アーキテクチャの研究に従事．