Network-on-Chip における Fat H-Tree トポロジに関する研究

松谷宏紀[†] 鯉渕道紘[††] 天野英晴[†]

Fat H-Tree は Fat Tree の代替として提案されたツリー型トポロジであり，2 個の H-Tree からなるトーラス構造を内包している．本論文では，Fat H-Tree 向けのルーティングアルゴリズムを提案し，Fat H-Tree トポロジの性能およびハードウェア量を他のツリー型トポロジと比較する．さらに，3 次元構造を持った IC 向けに Fat H-Tree のレイアウト方法を提案し，Fat H-Tree の 2 次元および 3 次元レイアウトについて配線量と消費エネルギーを評価する．その結果，1) Fat H-Tree は同規模の Fat Tree よりも面積性能比が高い，2) Fat H-Tree の消費エネルギーは同 Fat Tree より小さい，3) Fat H-Tree に要する配線資源は同 Fat Tree より若干多いが，現状のプロセス技術においては十分に実装できる，4) Fat H-Tree の総配線長は本論文で提案した 3 次元レイアウトによって最大 49.8%削減できる，という点を確かめた．

The Study of Fat H-tree Topology for Network-on-chips

Hiroki Matsutani,[†] Michihiro Koibuchi[††] and Hideharu Amano[†]

Fat H-Tree is a tree-based interconnection network providing a torus structure, which is formed by combining two folded H-Tree networks, and is an attractive alternative to tree-based networks such as Fat Trees. In this paper, new routing algorithms are proposed for Fat H-Tree, and performance and network logic area of Fat H-Tree are compared with those of other trees-based networks. In addition, an efficient layout of Fat H-Tree is proposed for three-dimensional ICs, and the two- and three-dimensional layout of Fat H-Tree are evaluated in terms of required wire resources and energy consumption. The results show that 1) Fat H-Tree outperforms a same-scale Fat Tree in terms of cost and performance; 2) Fat H-Tree consumes less energy than the Fat Tree; 3) Fat H-Tree slightly increases wire resources compared with the Fat Tree, but the current process technology can provide sufficient wire resources for implementing Fat H-Tree based on-chip networks; 4) The three-dimensional layout of Fat H-Tree reduces its total wire length by up to 49.8%.

1. はじめに

半導体技術の進歩によって単一チップ上にプロセッサやメモリ，I/O など複数の設計モジュールをタイル状に実装できるようになり，このようなタイルどうしの結合に Network-on-Chip（NoC）[1),2)] が用いられるようになった．さらに，最近では，チップの 3 次元実装[3)~5)] が注目されている．複数枚のウェハまたはダイを垂直方向に重ね合わせることで個々のチップサイズを小型化できるため，実装面積や配線長の削減が期待されている．このような状況を背景に，2005 年頃より 3 次元 IC 向けの NoC の研究が報告されている[6)~8)]．

NoC のトポロジは，2 次元，3 次元を問わず，アプリケーションの性能と面積，消費電力を決定付ける一要素である．NoC は高性能計算機以外にも，高いコスト効率が求められる組み込み向けチップなどで利用されることが多く，このような用途では，オンチップルータの面積が増えると，アプリケーションを実行する計算コアの面積が圧迫されるので，ルータの面積は極力小さくする必要がある．一方，ピン数によってデータ幅が制限されるチップ間結合網とは異なり，NoC では配線資源が豊富である．このような豊富な配線資源を効率的に利用できるトポロジが NoC において求められている[1)]．

NoC のトポロジとして，メッシュやトーラスなどのグリッド型トポロジのほかにツリー型トポロジが広範囲に利用されている．グリッド型とツリー型の網羅的な比較は本論文の範囲を超えるが，ツリー型トポロジにおいてもメッシュと同等の面積性能比を実現できると報告されている[9)]．そこで，本論文では，ツリー型トポロジを対象にチップ内の豊富な配線資源を効率

† 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University
†† 国立情報学研究所
National Institute of Informatics

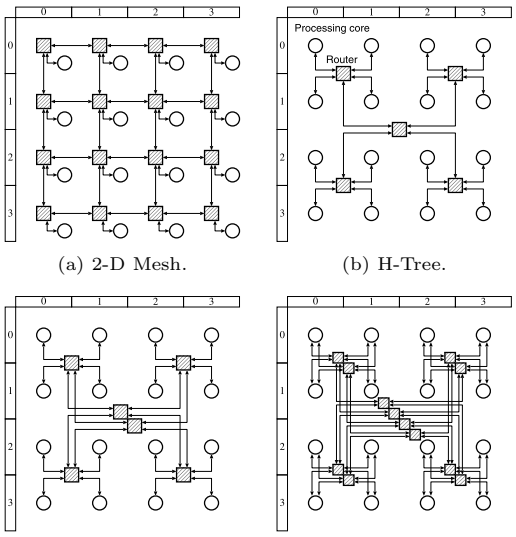
良く利用できる NoC トポロジについて検討する．

図 1 に，NoC における代表的なネットワークトポロジを示す．図中の四角はルータ，丸は計算コアをそれぞれ表す．

単純な 4 進ツリーを 2 次元レイアウトした H-Tree（図 1 (b)）は，トラフィックがツリーのルート付近に集中しやすい．そこで，Fat Tree ではツリーを多重化することでツリーのルート付近のトラフィックの集中を緩和する．Fat Tree には様々な形態があり，ここでは Fat Tree を（上向きリンク数 p，下向きリンク数 q，コアの上向きリンク数 c）の組で表記する．たとえば，図 1 (d) において各ルータは 2 本の上位リンク，4 本の下位リンク，各コアは 2 本の上位リンクを持つため，以後これを Fat Tree (2,4,2) と表記する．

一方，我々はリコンフィギャラブルプロセッサアレイ向けに Fat H-Tree と呼ばれるトーラス構造を内包したツリーを提案しており[10]，現在，この Fat H-Tree を NoC における既存のツリー型トポロジの代替として用いることを検討している．しかし，文献 10) では，Fat H-Tree の提案に重きが置かれており，Fat H-Tree 用の最短経路ルーティングは提案されていない．さらに，性能，結合網のハードウェア量，配線量，消費電力についての網羅的な評価も行われていない．加えて，ツリー型トポロジについては，3 次元 IC に対するレイアウトおよび評価は行われていない．

そこで，本論文では Fat H-Tree 用の最短経路ルーティングを提案し，実アプリケーションに基づいたシミュレーションによって性能を測定，Fat H-Tree ベースの NoC を RTL 設計して結合網のハードウェア量を見積もる．さらに，Fat H-Tree の 3 次元 IC 向けレイアウトを提案し，Fat H-Tree の 2 次元および 3 次元レイアウトの配線量についても評価する．まず 2 章で Fat H-Tree トポロジを説明する．次に 3 章で Fat H-Tree のルーティングを提案し，4 章で Fat H-Tree の 3 次元レイアウトを提案する．5 章において Fat H-Tree と既存のトポロジについて理論的に解析し，これらの解析結果を 6 章の評価で検証する．最後に 7 章で本論文をまとめる．



(a) 2-D Mesh. (b) H-Tree.

(c) Fat Tree (2,4,1). (d) Fat Tree (2,4,2).

図 1 代表的なネットワークトポロジ（16 コア）
Fig. 1 Typical network topologies (16-core).

2. Fat H-Tree トポロジ

Fat H-Tree は red tree と black tree と呼ばれる 2 つの H-Tree を組み合わせたトポロジであり，トーラス構造を内包している．各計算コアのネットワークインターフェイス（NI）は 2 つのポートを持ち，1 つを red tree との接続用に，もう片方を black tree との接続用に用いる．Fat H-Tree のフォーマルな定義は文献 10) に示されている．

a) Red Tree の定義

図 2 に red tree の例を示す．図 1 同様，図中の四角はルータ，丸は計算コアをそれぞれ表す．また，四角の中の数字は各ルータのランク数☆を表す．H-Tree のコア数を $2^n \times 2^n$ とするとき，この H-Tree は $\log_4(4^n) = n$ 個の階層を持つことになる．このとき，ルートに位置

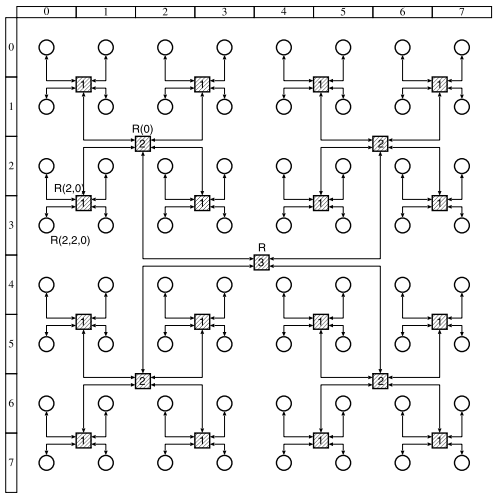


図 2 Fat H-Tree（red tree）
Fig. 2 Fat H-Tree (red tree).

☆ ツリーのルートを最上位，リーフを最下位とするとき，ルータのランク数はそのルータがツリーのどの階層に属しているかを表す．

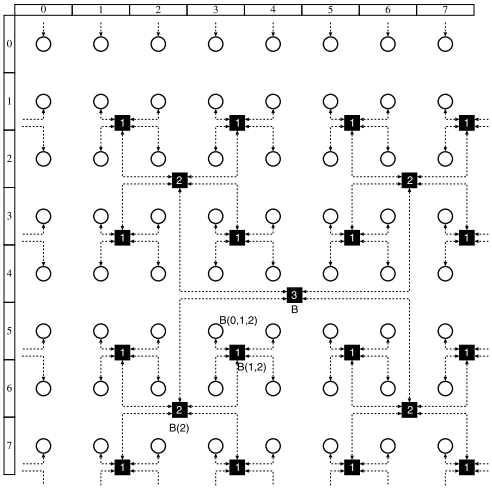


図 3 Fat H-Tree（black tree）
Fig. 3 Fat H-Tree (black tree).

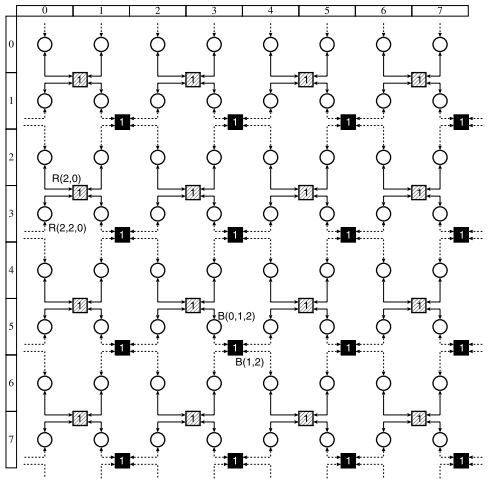


図 4 Fat H-Tree（rank 2 以上のルータは省略）
Fig. 4 Fat H-Tree (rank-2 or upper routers are not shown).

するルータ（最上位ルータ）のランク数は n である．

丸で描かれた計算コアには 2 次元座標 (x_{2D}, y_{2D}) がそれぞれ割り当てられる．以降，計算コアを rank 0 ルータと呼ぶ．まず，rank 0 ルータに対し次式で求まる **red tree 座標** $R(r_0, r_1, \ldots, r_{n-1})$ を割り当てる．

$$r_i = ((x_{2D}/2^i) \bmod 2) + 2 \times ((y_{2D}/2^i) \bmod 2) \quad (1)$$

red tree 座標 $R(r_0, r_1, \ldots, r_{n-1})$ となる rank 0 ルータの上位ルータを rank 1 ルータと呼び，$R(r_1, \ldots, r_{n-1})$ の red tree 座標を割り当てる．同様の方法で rank 2 ルータから rank n ルータにも red tree 座標を割り振る．ただし，最上位ランク（rank n）のルータの red tree 座標は R とする．例として図 2 に R, $R(0)$, $R(2,0)$, $R(2,2,0)$ の座標を示す．

b) Black Tree の定義

図 3 に black tree の例を示す．black tree は red tree の位置を右斜め下方向に 1 つずらしたものであり，rank 0 ルータの **black tree 座標** $B(b_0, b_1, \ldots, b_{n-1})$ は次の式で求まる．

$$b_i = ((((x_{2D} - 1) \bmod 2^n)/2^i) \bmod 2) + 2 \times ((((y_{2D} - 1) \bmod 2^n)/2^i) \bmod 2) \quad (2)$$

black tree においても rank 1 ルータから rank n ルータに black tree 座標を割り当てていく．最上位ランク（rank n）ルータの black tree 座標は B とする．

c) Fat H-Tree の定義

それぞれの計算コア（rank 0 ルータ）に対し，red tree および black tree を接続したものを **Fat H-Tree** と呼ぶ．

Fat H-Tree においては，rank 0 ルータと red tree および black tree の rank 1 ルータによってトーラス構造が形成される．Fat H-Tree が内包するトーラス構造を**図 4** に示す．

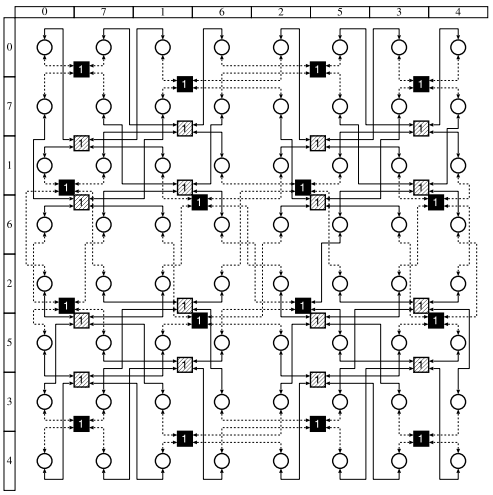


図 5 Folded Fat H-Tree（rank 2 以上のルータは省略）
Fig. 5 Folded Fat H-Tree (rank-2 or upper routers are not shown).

d) 2 次元レイアウト

Fat H-Tree は，トーラス同様，畳み込むことで 2 次元チップ上に容易にレイアウトできる．**図 5** に Fat H-Tree の 2 次元レイアウトを示す．図 4 と図 5 は等価なトポロジである．

3. Fat H-Tree のルーティングアルゴリズム

まず，3.1 節で Fat H-Tree のルーティングアルゴリズムを提案する．3.2 節では，これらのルーティングをより少ない数の仮想チャネルで実現するための方

法を述べる．

3.1 基本アルゴリズム

文献 10) で提案されたルーティングは最短経路ルーティングではない．これを最短経路ルーティングになるように定義し直したものが以下に示す Dual Tree Routing（DTR）である．ここでは，DTR を含め 3 種のルーティングを提案する．

- **Single Tree Routing（STR）：** パケットの送信元において，毎回 red tree または black tree のどちらか一方を選び，H-Tree と同じ方法でルーティングする．つまり，STR ではリーフ方向（down）からルート方向（up）へのチャネル間の転送パターンが禁止される．以後，このようなチャネル間の禁止転送パターンを**禁止ターン**と呼ぶ．
- **Dual Tree Routing（DTR）：** 任意の最短経路を利用するために DTR では禁止ターンを設定しない．そのため，経路の途中でツリーを切り替えることができる．ただし，ツリー間の循環依存を取り除くために，red tree から black tree に乗り換えるときに仮想チャネルの番号をインクリメントする．文献 10) で提案されたルーティングは，ツリーの切替え可能な回数が制限された DTR であるといえる．
- **Torus Routing（TOR）：** rank 0 ルータおよび rank 1 ルータによって形成されるトーラス構造（図 4）のみを用いてルーティングする．そのため，TOR では rank 1 ルータから rank 2 ルータへの移動が禁止ターンとなる．DTR 同様，ツリーの切替え時に仮想チャネル番号をインクリメントする．

上記のうち DTR のみが最短経路ルーティングである．

各ルーティングにおける経路設定は次の手順で行う．

(1) 使用するルーティングに応じた禁止ターンをセットする．
(2) 禁止ターンに違反しない経路集合のうち，ホップ数が最短となるものをダイクストラのアルゴリズムを用いて探索する．
(3) 探索された経路集合をルーティングテーブルに格納する．

定理 STR において 1 本，DTR および TOR において $(\lfloor H_{max}/4 \rfloor + 1)$ 本の仮想チャネルを用いる場合，各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．ただし，H_{max} を各ルーティングにおける最大ホップ数とする．

証明 STR では，循環構造を持たない単一ツリー内でパケット転送が完結するためデッドロックは起きない．一方，DTR と TOR においては，1) ツリー内で循環は発生しない，2) red tree から black tree への切替え時に仮想チャネル番号をインクリメントするためツリー間においても循環は生じない，という 2 点によりデッドロックフリーが保証される． ∎

16 コアの Fat H-Tree において，DTR および TOR の H_{max} は 4 である．したがって，それぞれ 2 本の仮想チャネルが必要になる．一方，64 コアの Fat H-Tree では DTR の H_{max} は 6，TOR の H_{max} は 8 である．したがって，DTR は 2 本，TOR は 3 本の仮想チャネルが必要になる．

3.2 仮想チャネル数の削減手法

前節で述べたとおり，DTR と TOR ではネットワークの規模に応じて仮想チャネルが必要になる．しかし，システムによってはルータが仮想チャネル機構をサポートしていない場合や，仮想チャネルを持っていても仮想チャネル数が足りなくなる事態が起こりうる．DTR および TOR で必要な仮想チャネル数を削減するための方法を以下に 2 つあげる．

- **Eject-and-reinjection：** 循環依存を引き起こすパケットを中継コアで 1 度受信した後，この中継コアから本来の宛先に向けてパケットを再注入する[11),12)]．
- **Deadlock-recovery：** ルータにデッドロック検出・回復機構を持たせる．ルータによってデッドロックが検出されると，循環依存を引き起こしたパケットを廃棄して循環依存を取り除く[13)]．

eject-and-reinjection では，パケットを中継コアのバッファ領域に一時的に受信することで，中継コアの通過前と後で循環依存を断ち切る．中継コアによるパケットの受信および再注入によって通信遅延が増加するが，パケットの中継回数が少なければその影響は小さい．たとえば，64 コアの Fat H-Tree において TOR は本来 3 本の仮想チャネルを必要とするが，3 本目の仮想チャネルへの切替えが生じる直前のコアにおいて eject-and-reinjection を実行した場合，必要な仮想チャネル数を 2 本に抑えることができる．6.4 節のシミュレーションでは，64 コアの TOR において必要な仮想チャネル数を 2 本に抑えるために一部のパケットに eject-and-reinjection を適用しているが，大半のパケットは eject-and-reinjection されずに転送される．

一方，deadlock-recovery は並列計算機向けに広く研究されているものの，ルータにデッドロック検出・

回復機構が必要となるため，現状の NoC ではほとんど利用されていない．

3.3 固定型ルーティングへの変換手法

ルーティングアルゴリズムは，パケットごとに動的に通信経路が選択される適応型ルーティングと，静的に経路が決まる固定型ルーティングに大別される．本章で提案したルーティングはすべて複数の代替経路が利用できるため，適応型ルーティングとしても，固定型ルーティングとしても利用できる．

このようなルーティングアルゴリズムを固定型ルーティングとして用いる場合，複数の代替経路の中から 1 つの経路を選択する経路選択アルゴリズム[14),15)] が必要になる．代表的な経路選択アルゴリズムとして，1) ランダムに経路を選択するもの，2) トラフィックの静的分析に基づき経路の集中を防ぐもの[14)] が知られている．6.4 節のシミュレーションでは，DTR および TOR を固定型ルーティング化するために文献 14) のアルゴリズムを採用した．

4. 3 次元 IC 向けレイアウト

Fat Tree および Fat H-Tree ではツリーの多重化により多数のリンクを持つため，ネットワークの規模が大きくなると配線量が増加しやすい．そこで，本論文では単一のツリーを複数のレイヤに分割し，これらを 3 次元 IC 上にレイアウトすることで総リンク長の削減を試みる．本章ではこのためのレイアウト方法を提案する．

3 次元 IC には，ダイどうしをワイヤボンディングで結合するもの，マイクロバンプで結合するもの，誘導結合するもの，また，ウェハどうしを貫通ビアで結合するものなどがある[5)]．本論文では 3 次元 IC を構成する各ダイまたはウェハを文献 5) にならい **tier** と呼ぶ．ここでは tier 間リンクを最も高密度に実装できると期待されている貫通ビアを想定して議論を進める．文献 5) によると，この貫通ビアによる方法では，tier 間の間隔は 5 μm〜50 μm 程度まで抑えることができるため，垂直方向のリンク長は水平方向のリンクに比べて無視できるほど短くなる．また，貫通ビアのサイズは 1 μm 角〜10 μm 角程度まで小型化できると報告されている[4),5),8)]．

本章では，まず Fat Tree の 3 次元レイアウトを示し，次に Fat H-Tree の 3 次元レイアウトを提案する．これらの 3 次元レイアウトと既存の 2 次元レイアウトをもとに，6 章では配線量と消費エネルギーについて評価する．

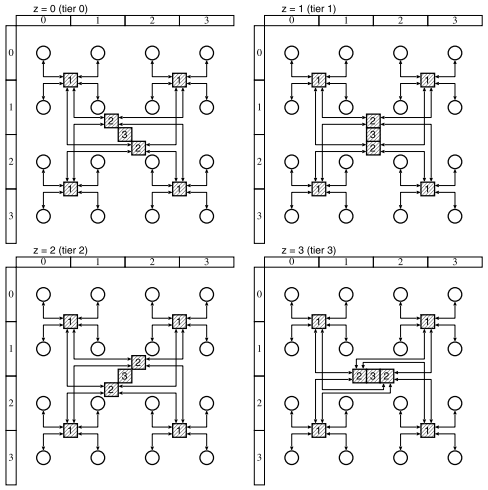


図 6 64 コア Fat Tree (2, 4, 1) の 4 分割
Fig. 6 Splitting a 64-core Fat Tree (2, 4, 1) into four.

4.1 Fat Tree の 3 次元レイアウト

Fat Tree の 4 分割

Fat Tree の 2 次元レイアウトを，半分の寸法からなる 4 枚の tier に分割し 3 次元的に結合する．これによって，最上位リンクは長いところでも距離数十 μm 以下の垂直リンクに置き換えられ，配線長は大幅に削減される．

Fat Tree の 4 分割手順は次のとおりである．

(1) （チップの分割）コア数を $2^n \times 2^n$ 個とするとき，各コアの 2 次元座標 (x_{2D}, y_{2D}) を，次のような 3 次元座標 (x_{3D}, y_{3D}, z_{3D}) に変換する．

$$x_{3D} = x_{2D} \bmod 2^{n-1}$$
$$y_{3D} = y_{2D} \bmod 2^{n-1}$$
$$z_{3D} = 2 \times \lfloor y_{2D}/2^{n-1} \rfloor + \lfloor x_{2D}/2^{n-1} \rfloor$$

たとえば，64 コアの Fat Tree (2, 4, 1) は，**図 6** のように 4 枚の tier（それぞれは 16 コア）に分割される．

(2) （ルータの配置）既存の 2 次元レイアウトをもとに，ルータを各階層に均等に振り分ける．図 6 の例では，各階層は 4 個の rank 1 ルータ，2 個の rank 2 ルータ，1 個の rank 3 ルータを持つ．

(3) （垂直リンクの配置）他階層に配置されたルータどうしをつなぐ最上位リンクは，貫通ビアなどの結合方式[5)] を用いて結合される．

(4) （ルータの再配置）垂直リンクが 1 カ所に集中しないようにルータの位置を調整する．図 6 のように，各階層の中心に rank 3 ルータを置き，その周囲に他の階層と重ならないように rank 2 ルータを密接に配置する．

これにより，最上位リンクは垂直リンクまたは同一

tier 内のごく短い配線に置き換えられ，配線長は大幅に削減される．なお，この方法は H-Tree や (2,4,2) など他の構成の Fat Tree に対しても適用できる．

Fat Tree の 2^i 分割

i を正の整数とするとき，Fat Tree の 2^i 分割は 4 分割と 2 分割を組み合わせることで実現できる．2 分割の場合は，2 次元のコア座標から 3 次元座標への変換は次のようになる．

$$x_{3D} = x_{2D}$$
$$y_{3D} = y_{2D} \bmod 2^{n-1}$$
$$z_{3D} = \lfloor y_{2D}/2^{n-1} \rfloor$$

これ以外の手順は 4 分割の場合と同じである．

4.2 Fat H-Tree の 3 次元レイアウト

Fat H-Tree はトーラス構造を内包するため，3 次元空間にレイアウトする際もこれが保存されていなければならない．

Fat H-Tree の 4 分割

Fat H-Tree の 4 分割手順は次のとおりである．

$$x_{3D} = \begin{cases} x_{2D} \bmod 2^{n-1} & x_{2D} < 2^{n-1} \\ 2^{n-1} - (x_{2D} \bmod 2^{n-1}) & x_{2D} \geq 2^{n-1} \end{cases}$$

$$y_{3D} = \begin{cases} y_{2D} \bmod 2^{n-1} & y_{2D} < 2^{n-1} \\ 2^{n-1} - (y_{2D} \bmod 2^{n-1}) & y_{2D} \geq 2^{n-1} \end{cases}$$

$$z_{3D} = 2 \times \lfloor y_{2D}/2^{n-1} \rfloor + \lfloor x_{2D}/2^{n-1} \rfloor$$

(1) （チップの分割）各コアの 2 次元座標 (x_{2D}, y_{2D}) を，上記のような 3 次元座標 (x_{3D}, y_{3D}, z_{3D}) に変換する．これにより，元の 2 次元レイアウトはチップの中心を軸に水平方向，および，垂直方向に折り畳まれ 4 枚に分割される．たとえば，図 2 の red tree は**図 7** のように，図 3 の black tree は**図 8** のように分割される．

(2) （ルータの配置）ルータを各階層に均等に振り分ける．この例では，red tree および black tree ともに，各階層は 4 個の rank 1 ルータ，1 個の rank 2 ルータを持つ．一方，rank 3 ルータについては，red tree は tier 1 に，black tree は tier 2 にそれぞれ 1 個ずつ持つ．

(3) （垂直リンクの配置）他階層に配置されたルータどうしをつなぐ．図 8 の black tree において「to tier n」と記されたリンクは tier n の rank 1 ルータへとつながる．

図 7 および図 8 を組み合わせたものが Fat H-Tree の 3 次元レイアウトである．

この 3 次元レイアウトにおいて分割後もトーラス（リング）構造が保存されている一例として，コア $(1,0,0)$ を起点に $y+$ 方向に移動し続けるとリング構造を一周して再びコア $(1,0,0)$ にたどり着くことを示す．図 7 の点線矢印 1 が示すように，tier 0 に配置されたコア $(1,0,0)$ の $y+$ 方向 2 ホップ先はコア $(1,1,0)$ である．その次は図 8 の点線矢印 2 が示すようにコア $(1,2,0)$，図 7 の点線矢印 3 が示すようにコア $(1,3,0)$ へと移動した後，点線矢印 4 のように tier 2 のコア $(1,3,2)$ に移る．tier 2 においても同様に，コア $(1,3,2)$，コア $(1,2,2)$，コア $(1,1,2)$，コア $(1,0,2)$ を経て，点線矢印 8 が示すように再び tier 0 のコア $(1,0,0)$ に戻る．

Fat H-Tree の 3 次元レイアウトは，元の 2 次元レイアウトを分割したい枚数分だけ折り畳むことで得られる．このような折り畳みによる 2 次元から 3 次元レ

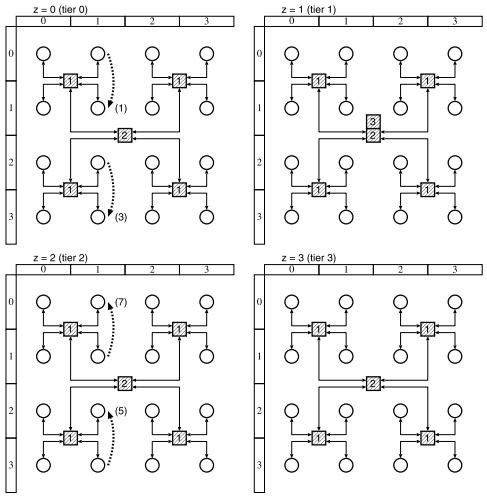


図 7 64 コア Fat H-Tree の 4 分割（red tree）
Fig. 7 Splitting a 64-core red tree into four.

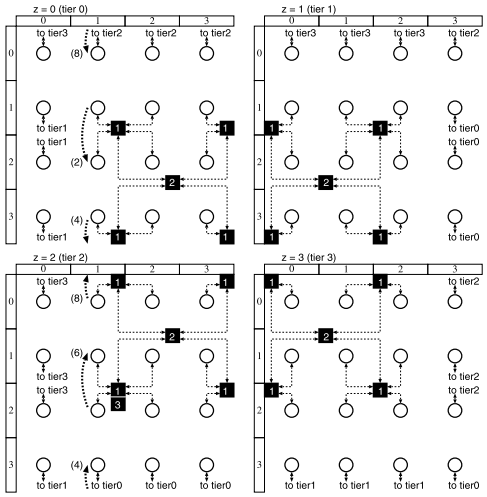


図 8 64 コア Fat H-Tree の 4 分割（black tree）
Fig. 8 Splitting a 64-core black tree into four.

イアウトへの変換は，文献 16) においても検討されており，少ない貫通ビア数で長い配線を短くできると考えられている．

Fat H-Tree の 2^i 分割

Fat H-Tree の 2 分割では，2 次元のコア座標から 3 次元座標への変換は次のようになる．

$$x_{3D} = x_{2D}$$
$$y_{3D} = \begin{cases} y_{2D} \bmod 2^{n-1} & y_{2D} < 2^{n-1} \\ 2^{n-1} - (y_{2D} \bmod 2^{n-1}) & y_{2D} \geq 2^{n-1} \end{cases}$$
$$z_{3D} = \lfloor y_{2D} / 2^{n-1} \rfloor$$

このとき，y 方向のリング（トーラス構造）は 2 枚の tier にまたがって形成されるが，x 方向のリングは単一 tier 内で完結させる必要がある．よって，x 方向に対してのみ図 5 のように畳み込みを行う．これ以外の手順は 4 分割と同じである．

5. Fat H-Tree トポロジの解析

5.1 スループットの理想値

スループットの理想値 Θ_{ideal} は次の関係式で表される[13]．

$$\Theta_{ideal} \leq \frac{2bB_c}{N} \quad (3)$$

ただし，N をコア数，b をチャネルあたりの帯域，B_c を channel bisection とする．

表 1 に各トポロジの B_c を示す．表中の HT は H-Tree，FT1 は Fat Tree (2,4,1)，FT2 は Fat Tree (2,4,2)，FHT は Fat H-Tree を表す．コア数 $N = 2^n \times 2^n$ とするとき，ツリーのランク数は $\log_4(N) = \log_4(4^n) = n$ である．

コア数 N の Fat H-Tree における B_c は $2^{n+2} + 8$ となる．この第 2 項は H-Tree 2 つ分（red tree と black tree）の channel bisection であり，第 1 項はルータ–コア間チャネルによって得られる channel bisection である．Fat H-Tree ではルータ–コア間チャネルによって形成されるトーラス構造により，単に H-Tree を二重化する場合に比べて飛躍的に Θ_{ideal} が向上する．Fat H-Tree の実スループットは，6 章でフリットレベルシミュレータを用いて確認する．

5.2 平均ホップ数

N コアのネットワークにおいて，有効なソース–ディスティネーション・ペアは $(N^2 - N)$ 個存在するため，平均ホップ数 H_{ave} は次の式で計算できる．

$$H_{ave} = \frac{1}{N^2 - N} \sum_{x,y \in N} H_{(x,y)} \quad (4)$$

ただし，$H_{(x,y)}$ はコア x からコア y へのホップ数とする．Mesh や Torus などの直接網においては，ルータとコアが単一ノードとして実装されるため，コア–ルータ間リンクを 1-hop にカウントしないものとした．一方，Fat Tree や Fat H-Tree などの間接網においては，コアどうしはルータを介して間接的に接続されるため，コア–ルータ間リンクを 1-hop にカウントするものとした．

表 1 Channel bisection B_c
Table 1 Channel bisection B_c.

	N-core	16-core	64-core	256-core
HT	4	4	4	4
FT1	2^{n+1}	8	16	32
FT2	2^{n+2}	16	32	64
FHT	$2^{n+2} + 8$	24	40	72
Mesh	2^{n+1}	8	16	32
Torus	2^{n+2}	16	32	64

表 2 平均ホップ数 H_{ave}
Table 2 Average hop count H_{ave}.

	Routing	16-core	64-core	256-core
HT,FT	up*/down*	3.60	5.43	7.36
FHT	STR	3.20	5.02	6.90
FHT	DTR	3.20	4.84	6.78
FHT	TOR	3.20	5.65	10.83
Mesh	DOR	2.67	5.33	10.67
Torus	DOR	2.13	4.06	8.03

表 2 にユニフォームトラフィックを用いた際の H_{ave} を示す．H_{ave} は使用するルーティングによって異なる．Mesh および Torus では次元順ルーティング (dimension-order routing, DOR)[13]，H-Tree と Fat Tree では up*/down*ルーティング☆を用いた．Fat H-Tree については STR，DTR，TOR の場合の結果をそれぞれ示す．表 2 より，DTR を用いた場合 Fat H-Tree の H_{ave} は Fat Tree より 7.9〜11.1%小さいことが分かる．

5.3 ルータの個数

ルータの個数は結合網の面積や実装コストに影響を与える．

ツリー型トポロジにおいてルータの下向きリンク数 q が 4 のとき，ランク数 n の H-Tree におけるルータの個数 R_{ht} は

$$R_{ht} = (q^n - 1) \,/\, (q - 1) = (4^n - 1) \,/\, 3 \quad (5)$$

となり，Fat Tree (2,4,1) におけるルータ数 R_{ft1} は

$$R_{ft1} = (q^n - 2^n) \,/\, (q - 2) = (4^n - 2^n) \,/\, 2 \quad (6)$$

となる．Fat Tree (2,4,2) は Fat Tree (2,4,1) 2 個分

☆ 本論文では，ツリーのリーフ方向（down）からルート方向（up）へのターンを禁止するルーティングを up*/down*ルーティングと呼ぶ．

表 3 ルータの個数 R
Table 3 Number of routers R.

	N-core	16-core	64-core	256-core
HT	$(4^n-1)/3$	5	21	85
FT1	$(4^n-2^n)/2$	6	28	120
FT2	4^n-2^n	12	56	240
FHT	$2(4^n-1)/3$	10	42	170
Mesh	N	16	64	256
Torus	N	16	64	256

のルータを，Fat H-Tree は H-Tree 2 個分のルータを必要とする．

以上をまとめた結果を**表 3** に示す．これまで見てきたように，Fat H-Tree は Fat Tree (2,4,2) より channel bisection および平均ホップ数の点で有利であるにもかかわらず，Fat H-Tree のルータの個数は Fat Tree (2,4,2) より少なく済んでいる．ただし，この見積りは NI のコストを考慮していない．実際，Fat Tree (2,4,2) と Fat H-Tree の NI はルータ 2 個と接続するために 2 ポート構成となるうえに，Fat H-Tree の NI は片方のポートからもう片方のポートへのパケット転送機能が必要となり面積がさらに増える．6 章では，NI を含め NoC 全体を論理合成して実際のハードウェア量を測定する．

5.4 結合網の総リンク長

本節では，隣接コア間距離を 1-unit として，各トポロジの総リンク長を見積もる．まず，通常の 2 次元レイアウトの総リンク長を計算し，次に 3 次元レイアウトとしてこのチップを 4 枚の tier に分割したときの総リンク長を示す．つまり，16 コアのネットワークであれば 2×2 コアの tier が 4 枚，64 コアは 4×4 コアの tier が 4 枚，256 コアは 8×8 コアの tier が 4 枚に分割されることになる．

2 次元レイアウト

ランク数 n の H-Tree の総リンク長 $L_{2D,ht}$ を次式で表す．

$$L_{2D,ht}=\sum_{i=1}^{n} l_{ht}^{i}\cdot r_{ht}^{i} \tag{7}$$

ただし，l_{ht}^{i} を H-Tree における rank i ルータからその 4 個の下位ルータへの総リンク長，r_{ht}^{i} を rank i ルータの数とする．計算コア数を $N=2^n\times2^n$ とすると，$l_{ht}^{i}=2^{i+1}$，$r_{ht}^{i}=N/4^i$ となる．よって，式 (7) は次式のように変形できる．

$$L_{2D,ht}=\sum_{i=1}^{n}2^{i+1}\cdot\frac{N}{4^i}=2N\left(\frac{2^n-1}{2^n}\right) \tag{8}$$

Fat H-Tree は 2 つの畳み込まれた H-Tree からなる．H-Tree を畳み込むと各リンクの長さは 2 倍になるが，最上位リンクのみ長さが 1-unit に短縮される．これは，畳み込みによって 1-unit 四方の領域に 4 個の rank $(n-1)$ ルータと 1 個の rank n ルータを配置できるためである．したがって，Fat H-Tree の総リンク長 $L_{2D,fht}$ は次の式で計算できる．

$$L_{2D,fht}=8+8N\left(\frac{2^{n-1}-1}{2^{n-1}}\right) \tag{9}$$

2 次元レイアウトにおける各トポロジの総リンク長を「2D」と記して**表 4** の上半分にまとめる．64 コア以下のネットワークでは Fat H-Tree の総リンク長は Fat Tree (2,4,2) よりわずかに長いが，これは近年のプロセス技術において大きな差ではないと考えられる．これに関しては 6.2 節で検証する．

表 4 ネットワークの総リンク長 L（隣接コア間距離を 1-unit）
Table 4 Total unit-length of links L (1-unit = distance between neighboring two cores).

	N-core	16core	64core	256core
HT(2D)	式 (8)	24	112	480
FT1(2D)	nN	32	192	1,024
FT2(2D)	$2nN$	64	384	2,048
FHT(2D)	式 (9)	72	392	1,800
Mesh(2D)	$2(N-2^n)$	24	112	480
Torus(2D)	$4(N-2^n)$	48	224	960
HT(3D)	式 (10)	16	96	448
FT1(3D)	$(n-1)N$	16	128	768
FT2(3D)	$2(n-1)N$	32	256	1,536
FHT(3D)	式 (11)	40	200	904
Mesh(3D)	$2(N-2^{n+1})$	16	96	448
Torus(3D)	$4(N-2^{n+1})$	32	192	896

3 次元レイアウト

3 次元 IC では tier 間の距離は数十 μm 程度となるため[5]，垂直方向のリンクは水平方向に比べて圧倒的に短い．よって，ここでは垂直リンクの長さは考慮しないものとする．

まず，ランク数 n の H-Tree を 3 次元化したときの総配線長 $L_{3D,ht}$ を求める．ここでは 4 枚の tier に分割するため，最上位リンクは長さ 0 の垂直リンクまたは同一 tier 内のごく短い配線（これも長さ 0 と見なす）に置き換えられる．そのため，式 (8) は 3 次元化によって次式のように変化する．

$$L_{3D,ht}=\sum_{i=1}^{n-1}2^{i+1}\cdot\frac{N}{4^i}=2N\left(\frac{2^{n-1}-1}{2^{n-1}}\right) \tag{10}$$

Fat H-Tree は red tree と black tree に分けて考える．red tree は H-Tree の 3 次元レイアウトと等価である（図 7）．一方，black tree はこれに加え，最上位リンクの長さとしてそれぞれ 2-unit 必要となる（図 8）．

以上より，Fat H-Tree の 3 次元レイアウトにおける総リンク長は次式で計算できる．

$$L_{3D,fht} = 8 + 4N\left(\frac{2^{n-1}-1}{2^{n-1}}\right) \qquad (11)$$

3 次元レイアウトにおける総リンク長を「3D」と記して表 4 の下半分にまとめる．3 次元化によって，Fat H-Tree の総リンク長は 44.4〜49.8%，Fat Tree の総リンク長は 25.0〜50.0%削減された．Fat H-Tree および Fat Tree におけるこの削減率は，メッシュやトーラスにおける削減率よりも大きい．また，64 コア以上のネットワークでは Fat Tree より Fat H-Tree のほうが削減率が大きい．実際，64 コア以上のネットワークでは Fat H-Tree は Fat Tree (2,4,2) より総リンク長が 21.9%以上短くなった．

Fat H-Tree の 2 次元レイアウト（図 5）では，隣接コアを 1 個飛ばしに配置することでトーラス（リング）構造を 2 次元平面上に実装している．一方，Fat H-Tree の 3 次元レイアウト（図 7 および図 8）では，チップを折り畳むことでリングが形成されるため，隣接コアを 1 個飛ばしに配置する必要はない．これが，3 次元化した場合に Fat H-Tree の総リンク長が大きく減った理由である．

6. Fat H-Tree トポロジの評価

6.1 結合網のハードウェア量

Verilog-HDL で記述されたルータ回路と NI 回路を組み合わせることで対象トポロジを構築し，これを 0.18 μm スタンダードセルライブラリを用いて Synopsys 社の Design Compiler で合成することで各トポロジのゲート数を見積もる☆．

本評価では文献 17) の 32-bit wormhole ルータ回路を用いた．このルータ回路は 4 段のパイプラインステージから構成され，各ステージごとに 1-flit 分のバッファを持つ．仮想チャネル数は 2 本とした☆☆．仮想チャネルごとに独立したバッファを持たせているため，物理チャネルごとに合計 8-flit 分のバッファを持つ．

NI 回路として 2-flit 分の FIFO を入力側と出力側にそれぞれ持たせた．Fat H-Tree と Fat Tree (2,4,2) では 2 ポートの NI が必要になる．とくに Fat H-Tree では，片方のポートからもう片方のポートへパケットを転送する機能が必要であり，このために Fat H-Tree 用の NI 内にはクロスバスイッチおよび 2 本分の仮想チャネルが実装されている．

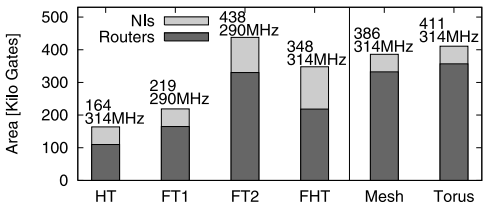


図 9 ルータと NI のゲート数（16 コア）
Fig. 9 Gate count of routers and NIs (16-core).

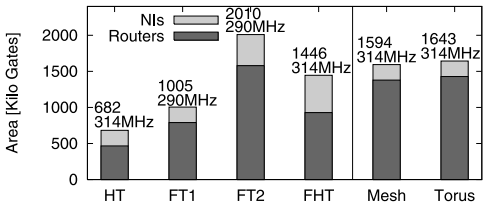


図 10 ルータと NI のゲート数（64 コア）
Fig. 10 Gate count of routers and NIs (64-core).

図 9 が 16 コア，**図 10** が 64 コアの合成結果である．Fat H-Tree の 2 ポート NI にはクロスバが内蔵されており，他のトポロジの NI よりゲート数が多い．それでも，Fat H-Tree のルータの個数は Fat Tree (2,4,2) よりも少ないため，結合網全体のゲート数は Fat Tree (2,4,2) より 20.5〜28.1%小さくなった．動作周波数に関しては大差は出ていない．

6.2 結合網の配線量

5.4 節で解析した各トポロジの総リンク長に基づき，Fat H-Tree に要す配線量について検討する．ここでは，12 mm 角のチップに 16 コアと 64 コアの NoC を 32-bit の双方向チャネルを用いて実装する場合を考える．このとき 1-unit の実際の長さは 16 コア構成で 3 mm，64 コアで 1.5 mm となる．

2 次元レイアウト

表 4 より，Fat H-Tree の配線長は 16 コア構成で 13.8 m，64 コアで 37.6 m となる．対して Fat Tree (2,4,2) の配線長は 16 コアで 12.3 m，64 コアで 36.9 m となる．

これらがチップ全体の配線資源に占める割合を概算する．最小配線ピッチが 0.1 μm の 0.18 μm プロセスを仮定し，ネットワークの配線には全配線層のうち 2 層分のみを使うものとする．12 mm 角のチップを想定しているので，1 配線層あたりのトラック数は 12,000

☆ 3.2 節で示した eject-and-reinjection を実行するには各計算コアに一定のバッファ領域が必要となるが，本評価ではこれらのバッファは計算コアがあらかじめ持っているものと仮定し，結合網のハードウェア量には含めない．

☆☆ Mesh における次元順ルーティングなど仮想チャネルを必要としないルーティングアルゴリズムもあるが，仮想チャネル機構はスループット，ルータのパイプライン構造自体に影響を与えるため，本論文ではすべてのトポロジにおいて仮想チャネル数を 2 本として比較評価を行った．

本，この配線層 2 層分の総配線長は 288 m となる☆．以上より，Fat H-Tree に要す配線資源は 16 コアで配線層 2 層分の 4.8%，64 コアで 13.1%，Fat Tree (2,4,2) の場合は 16 コアで 4.3%，64 コアで 12.8%であり，たかだか十数%以下である．実際には 6 層以上の配線層が利用できるため，Fat H-Tree は現状のプロセスで十分に実装可能であるといえる．しかも，プロセスの主流は 90 nm 以下に移り変わっており，配線資源の制約は今後ますます緩和される．

3 次元レイアウト

5.4 節で述べたとおり，Fat H-Tree の総リンク長は同 2 次元レイアウトに比べて 44.4〜49.8%短くなる．よって，配線資源の要求も同様に緩和される．

6.3 消費エネルギー

1-flit のデータを送信元から宛先コアに転送するとき，これに要する平均転送エネルギー E_{flit} は次式で計算できる．

$$E_{flit} = wH_{ave}(E_{sw} + E_{link}) \quad (12)$$

ただし，w を 1-flit のビット幅，H_{ave} を平均ホップ数，E_{sw} をルータまたは NI が 1-bit のデータ転送に消費するエネルギー，E_{link} をリンクが 1-bit のデータ転送に消費するエネルギーとする．

6.1 節で合成した仮想チャネルを 2 本持つルータおよび NI を，250 MHz での動作を仮定してゲートレベルでシミュレーションした．その結果，ルータの E_{sw} は 1.88 pJ，Fat H-Tree 以外の NI の E_{sw} は 1.27 pJ，Fat H-Tree の NI では 1.45 pJ となった．一方，E_{link} は次式で計算できる．

$$E_{link} = dV^2C_{wire}/2 \quad (13)$$

ただし，d を 1-hop あたりの平均距離，V を動作電圧，C_{wire} を配線容量とする．本評価では V を 1.8 V とし，C_{wire} は 0.18 μm プロセスを仮定するとき 414 fF/mm となった[19]．12 mm 角のチップを想定し，上記のパラメータをもとに，ツリーの 2 次元および 3 次元レイアウトにおける E_{flit} を計算した．16 コアの結果を**図 11**，64 コアの結果を**図 12** に示す．

2 次元レイアウト

グラフ中の薄い灰色の棒グラフが 2 次元レイアウトにおける E_{flit} である．参考までに 2-D Mesh と 2-D Torus の結果も載せてある．H-Tree と Fat Tree では，最上位リンクの利用率が高くデータの移動距離が長いため，転送エネルギーは Fat H-Tree よりも大きくなった．一方，Fat H-Tree では畳み込みによって 1-hop あたりの平均距離 d が伸びるが，平均ホップ数はこの中で最も小さく，トータルの転送エネルギーは Fat Tree より 6.7〜7.0%少なくて済んでいる．

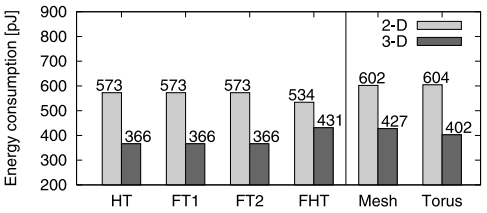


図 11 ツリーにおける転送エネルギー E_{flit}（16 コア）
Fig. 11 Energy consumption E_{flit} in trees (16-core).

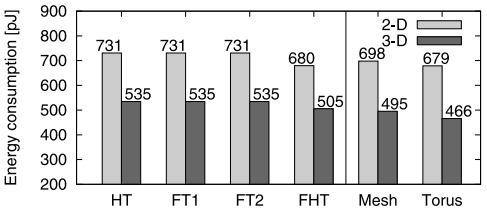


図 12 ツリーにおける転送エネルギー E_{flit}（64 コア）
Fig. 12 Energy consumption E_{flit} in trees (64-core).

3 次元レイアウト

次に，ツリーの 3 次元レイアウトにおける転送エネルギーを計算する．文献 5) によると貫通ビアの容量は 4.34 fF である．これは本評価で仮定している配線 10.5 μm 分の容量に相当し，水平方向の配線長に比べて十分に小さい．そのため，垂直リンクで消費されるエネルギーは考慮しないものとする．

グラフ中の濃い灰色の棒グラフが 3 次元レイアウトの E_{flit} である．2 次元レイアウトと比べて，Fat Tree の E_{flit} は最大 36.0%，Fat H-Tree の E_{flit} は最大 25.6%削減できた．以上より，今後普及が期待される 3 次元 IC において，4 章で提案した 3 次元レイアウトを用いることで，配線長および消費エネルギーを大幅に削減できることを確認した．

ただし，すでに多くの 3 次元 IC が実用化されているものの，3 次元化の問題点として，歩留まりの低下や放熱の難しさなどが指摘されている[5]．さらに，貫通ビアはウェハを貫通するように形成されるため，macro block や cell などが配置されていない領域に置く必要がある．しかも，貫通ビアのサイズは現在 1 μm 角〜10 μm 角程度と報告されており[4),5),8)]，これは水平方向の配線ピッチと比べて大きい．これらの問題が緩和されれば，本論文で提案した 3 次元レイアウトがより実用的になると考えられる．

☆ この見積りは配線資源の概算に有用であるが，チップ内には電源などの配線不可領域があるため，実際に利用可能な配線資源はこれよりも少ない[18]．

6.4 スループット

6.4.1 シミュレーション環境

各トポロジのスループットを測定するためにフリットレベルシミュレータを用いた．各ルータのスイッチング機構として，I/O バッファ，クロスバ，アービタを単純化したモデルを採用し，ヘッダフリットが隣接ルータまたは計算コアに転送されるのに 3 サイクルかかるものとする．パケット転送には wormhole 方式を用い，各チャネルは 1-flit 分のバッファを持つ．パケット長はヘッダ 1-flit 分を含め 16-flit とした．

ルーティングアルゴリズムとして，2-D Torus および Mesh には次元順ルーティング，Fat Tree には up*/down*ルーティング，Fat H-Tree には DTR と TOR を用いた．Fat Tree および Fat H-Tree のルーティングは適応型ルーティングとしても利用できるが，次元順ルーティングが固定型ルーティングであるので，本シミュレーションでは固定型ルーティングに統一した．Fat Tree および Fat H-Tree のルーティングを固定型ルーティングに変換する際には文献 14) の経路選択アルゴリズムを用い，トラフィックの集中をできる限り排除するようにした．

Torus および Fat H-Tree における構造上デッドロックを回避するため，仮想チャネルを 2 本用いた．なお，3.1 節で述べたとおり，64 コアの Fat H-Tree では TOR は本来 3 本の仮想チャネルを必要とするが，ここでは評価条件を揃えるために 3.2 節で述べた eject-and-reinjection を用いて TOR に必要な仮想チャネル数を 2 本に制限している．

シミュレーションに用いる通信パターンとして，ユニフォームトラフィックと NAS Parallel Benchmark（NPB）プログラムから得られた通信パターンを用いる．NPB の各プログラムは数値計算を扱う並列アプリケーションであるが，これらの通信パターンにはストリーム処理で頻出する fork/join に似た通信パターンが含まれる．今回，NPB から次のプログラムを用いる：Block Tridiagonal solver（BT），Scalar Penta-diagonal solver（SP），Conjugate Gradient（CG），Multi-Grid solver（MG），large Integer Sort（IS）．プログラムのクラスは “W” とし，計算コアの数は 16 と 64 とした．各トポロジの性能はアプリケーションのタスクマッピングによっても変化する．本評価では，トポロジとアプリケーションのすべての組合せごとに，平均ホップ数が最小になるようなタスクマッピングを探索し，評価に用いた．

6.4.2 シミュレーション結果

図 13 に 16 コアでユニフォームトラフィックを用いた際のスループット（accepted traffic）とレイテンシのグラフを示す．グラフ中の HT は H-Tree，FT1 は Fat Tree (2,4,1)，FT2 は Fat Tree (2,4,2)，FHTD は Fat H-Tree の DTR，FHTT は Fat H-Tree の TOR に対応する．括弧内に示してある平均ホップ数は表 2 の H_{ave} と同じである．スループットに関しては Torus，FHTT，FHTD，FT2，Mesh，FT1，HT の順となった．Fat H-Tree は 2-D Torus より高い channel bisection B_c を有するが，FHTD のスループットは Torus に劣っている．これは Fat H-Tree で最短経路のみを用いるとトラフィックがツリーのルート付近に集中してしまうためである．一方，FHTT のように rank 2 以上のリンクの使用を諦めれば，Fat H-Tree は Torus と等価となり，Torus に匹敵する性能が得られるが，非最短経路の導入によって平均ホップ数は増加する．

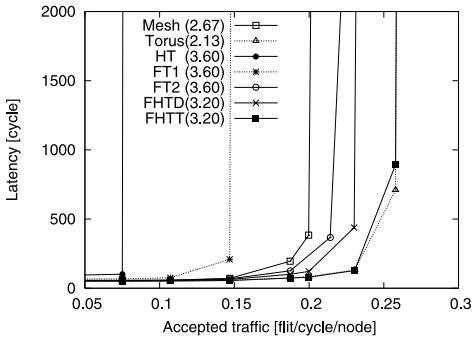


図 13 uniform トラフィック（16 コア）
Fig. 13 uniform traffic (16-core).

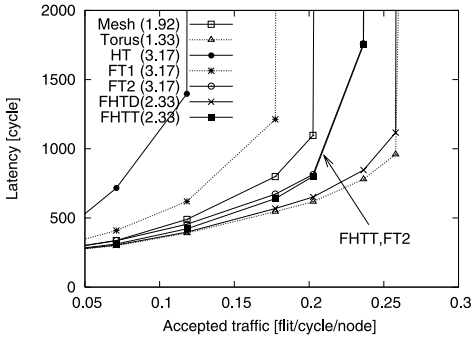


図 14 BT トラフィック（16 コア）
Fig. 14 BT traffic (16-core).

図 14，**図 15**，**図 16**，**図 17**，**図 18** に，16 コアにおける BT，SP，CG，MG，IS トラフィックのスループットとレイテンシを示す．CG を除き，FHTD は Torus に匹敵する性能を実現できている．なお，all-to-all 通信を含む IS ではユニフォームトラフィック同様，FHTT の性能が高いが，それ以外の比較的単純な通信では FHTT より FHTD のほうが有利なケース

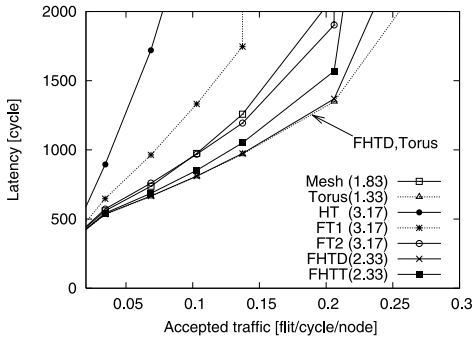


図 15 SP トラフィック（16 コア）
Fig. 15 SP traffic (16-core).

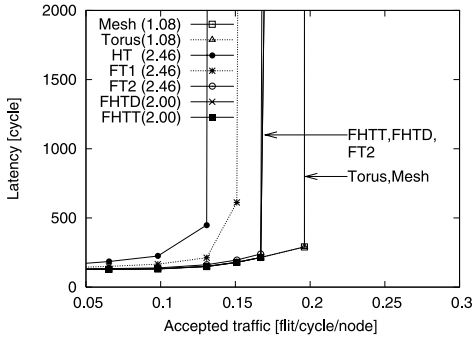


図 16 CG トラフィック（16 コア）
Fig. 16 CG traffic (16-core).

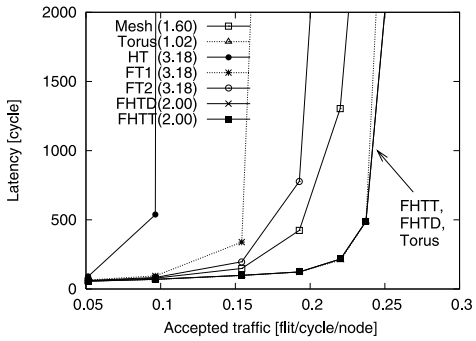


図 17 MG トラフィック（16 コア）
Fig. 17 MG traffic (16-core).

が目立つ．これは，単純な通信においては rank 2 以上のリンクを使用しても性能のボトルネックにはならず，むしろこれらの高位リンクが性能向上に寄与するためである．続いて 64 コアでの評価に移る．**図 19** に 64 コアでのユニフォームトラフィックの結果，**図 20**，**図 21**，**図 22**，**図 23**，**図 24** に NPB プログラムでの結果を示す．64 コアにおいても 16 コアと同様の傾向がみられた．

最後に Fat H-Tree と Fat Tree (2,4,2) の面積性能比について考察する．64 コアのユニフォームトラ

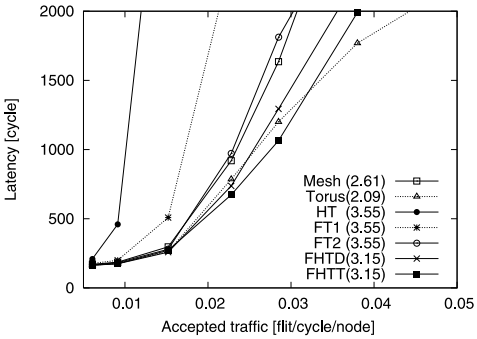


図 18 IS トラフィック（16 コア）
Fig. 18 IS traffic (16-core).

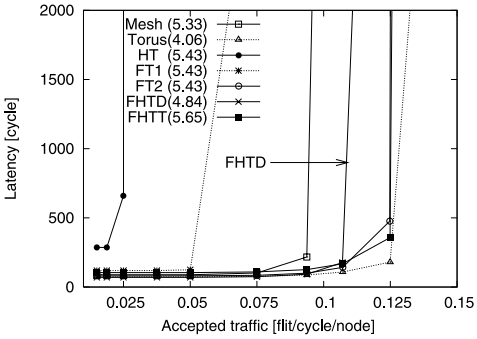


図 19 uniform トラフィック（64 コア）
Fig. 19 uniform traffic (64-core).

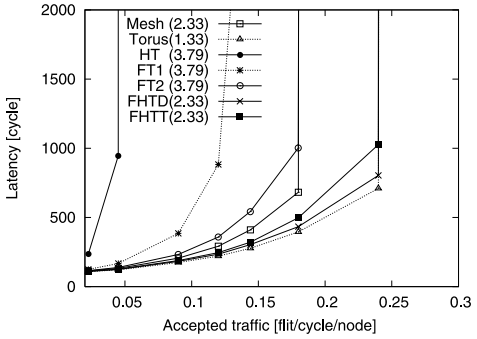


図 20 BT トラフィック（64 コア）
Fig. 20 BT traffic (64-core).

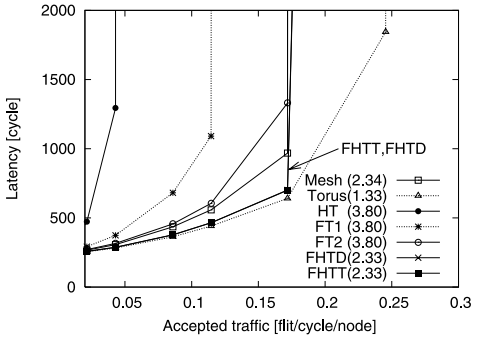


図 21 SP トラフィック（64 コア）
Fig. 21 SP traffic (64-core).

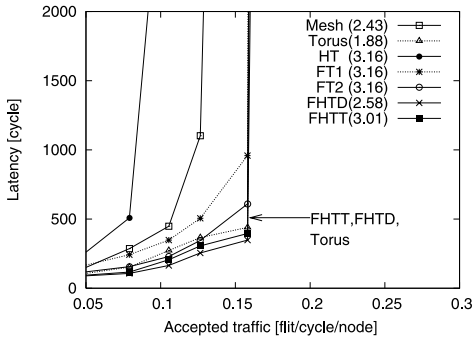


図 22 CG トラフィック（64 コア）
Fig. 22 CG traffic (64-core).

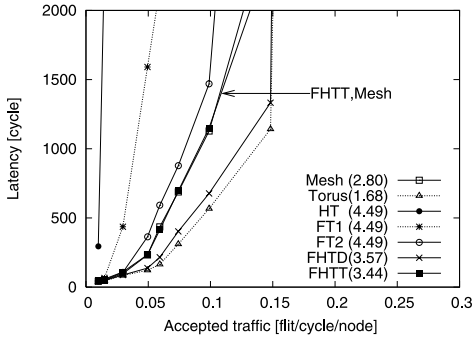


図 23 MG トラフィック（64 コア）
Fig. 23 MG traffic (64-core).

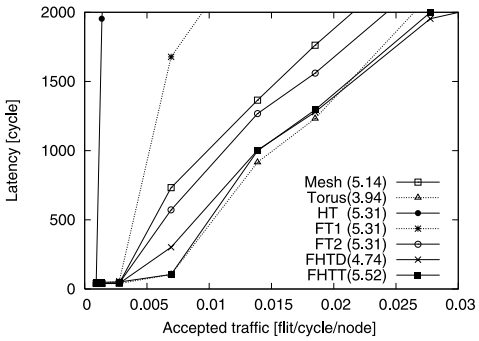


図 24 IS トラフィック（64 コア）
Fig. 24 IS traffic (64-core).

フィックにおいて FHTD がルート付近の混雑によって FT2 より劣っていることを除くと，Fat H-Tree は FT2 と同じかそれ以上の性能を実現できている．一方，6.1 節で示したとおり，Fat H-Tree のハードウェア量は Fat Tree (2,4,2) より 20.5〜28.1%小さい．したがって，Fat H-Tree は Fat Tree (2,4,2) より高い面積性能比を実現できているといえる．

7. ま と め

本論文では，Fat H-Tree トポロジの性能およびハードウェア量を他のツリー型トポロジと比較した．さらに，3 次元構造を持った IC 向けに Fat H-Tree のレイアウト方法を提案し，Fat H-Tree の 2 次元および 3 次元レイアウトについて配線量と消費エネルギーを評価した．その結果，1) Fat H-Tree の性能および平均ホップ数は 2 本の上位リンク，4 本の下位リンクを持つ Fat Tree より有利である，2) Fat H-Tree における結合網の面積は同 Fat Tree の面積より 20.5〜28.1%小さい，3) Fat H-Tree では平均ホップ数が小さいため，パケットの転送エネルギーについても同 Fat Tree より有利である，4) Fat H-Tree に要す配線資源は同 Fat Tree より若干多いが，現状のプロセス技術においては十分に実装できる，5) Fat H-Tree の総配線長は本論文で提案した 3 次元レイアウトによって最大 49.8%削減できる，ということが分かった．

謝辞 本研究の一部は，国立情報学研究所共同研究「ネットワークオンチップのアーキテクチャに関する研究」による．本研究は東京大学大規模集積システム設計教育研究センターを通し，シノプシス株式会社の協力で行われたものである．

参 考 文 献

1) Dally, W.J. and Towles, B.: Route Packets, Not Wires: On-Chip Interconnection Networks, *Proc. Design Automation Conference*, pp.684–689 (2001).
2) Benini, L. and Micheli, G.D.: *Networks on Chips: Technology And Tools*, Morgan Kaufmann (2006).
3) Black, B., Nelson, D.W., Webb, C. and Samra, N.: 3D Processing Technology and Its Impact on iA32 Microprocessors, *Proc. International Conference on Computer Design*, pp.316–318 (2004).
4) Das, S., Fan, A., Chen, K.-N., Tan, C.S., Checka, N. and Reif, R.: Technology, Performance, and Computer-Aided Design of Three-Dimensional Integrated Circuits, *Proc. International Symposium on Physical Design*, pp.108–115 (2004).
5) Davis, W.R., Wilson, J., Mick, S., Xu, J., Hua, H., Mineo, C., Sule, A.M., Steer, M. and Franzon, P.D.: Demystifying 3D ICs: The Pros and Cons of Going Vertical, *IEEE Design and Test of Computers*, Vol.22, No.6, pp.498–510 (2005).
6) Addo-Quaye, C.: Thermal-Aware Mapping and Placement for 3-D NoC Designs, *Proc. International System-on-Chip Conference*, pp.25–28 (2005).

7) Pavlidis, V.F. and Friedman, E.G.: 3-D Topologies for Networks-on-Chip, *Proc. International System-on-Chip Conference*, pp.285–288 (2006).
8) Li, F., Nicopoulos, C., Richardson, T., Xie, Y., Narayanan, V. and Kandemir, M.: Design and Management of 3D Chip Multiprocessors Using Network-in-Memory, *Proc. International Symposium on Computer Architecture*, pp.130–141 (2006).
9) Kapre, N., Mehta, N., deLorimier, M., Rubin, R., Barnor, H., Wilson, M.J., Wrighton, M. and DeHon, A.: Packet-Switched vs. Time-Multiplexed FPGA Overlay Networks, *Proc. Symposium on Field-Programmable Custom Computing Machines* (2006).
10) 山田 裕, 天野英晴, 鯉渕道紘, 上樂明也, 安生健一朗：リコンフィギャラブルプロセッサアレイ向けチップ内接続網：Fat H-Tree, 電子情報通信学会論文誌 D, Vol.J89-D, No.9, pp.1923–1934 (2006).
11) Flich, J., Lopez, P., Malumbres, M.P. and Duato, J.: Boosting the Performance of Myrinet Networks, *IEEE Trans. Parallel and Distributed Systems*, Vol.13, No.7, pp.693–709 (2002).
12) Matsutani, H., Koibuchi, M. and Amano, H.: Enforcing Dimension-Order Routing in On-Chip Torus Networks without Virtual Channels, *Proc. International Symposium on Parallel and Distributed Processing and Applications*, pp.207–218 (2006).
13) Dally, W.J. and Towles, B.: *Principles and Practices of Interconnection Networks*, Morgan Kaufmann (2004).
14) Sancho, J.C. and Robles, A.: Improving the Up*/Down* Routing Scheme for Networks of Workstations, *Proc. International Euro-Par Conference on Parallel Processing*, pp.882–889 (2000).
15) Koibuchi, M., Jouraku, A. and Amano, H.: Path Selection Algorithm: The Strategy for Designing Deterministic Routing from Alternative Paths, *PARALLEL COMPUTING*, Vol.31, No.1, pp.117–130 (2005).
16) Cong, J., Luo, G., Wei, J. and Zhang, Y.: Thermal-Aware 3D IC Placement Via Transformation, *Proc. Asia and South Pacific Design Automation Conference*, pp.780–785 (2007).
17) 松谷宏紀, 鯉渕道紘, 天野英晴：オンチップトーラス網における仮想チャネルフリールーティング, 情報処理学会論文誌：コンピューティングシステム, Vol.47, No.SIG 12, pp.12–24 (2006).
18) Dally, W.J. and Poulton, J.W.: *Digital Systems Engineering*, Cambridge University Press (1998).
19) Ho, R., Mai, K.W. and Horowitz, M.A.: The Future of Wires, *Proc. IEEE*, Vol.89, No.4, pp.490–504 (2001).

(平成 19 年 1 月 22 日受付)
(平成 19 年 5 月 17 日採録)

松谷 宏紀（学生会員）

平成 16 年慶應義塾大学環境情報学部卒業．平成 18 年同大学大学院理工学研究科開放環境科学専攻修士課程修了．現在，同大学院理工学研究科開放環境科学専攻博士課程．平成 18 年度より日本学術振興会特別研究員．チップ内ネットワークの研究に従事．

鯉渕 道紘（正会員）

平成 12 年慶應義塾大学理工学部情報工学科卒業．平成 15 年同大学大学院理工学研究科開放環境科学専攻博士課程修了．博士（工学）．平成 14 年度より日本学術振興会特別研究員．現在，国立情報学研究所助教，総合研究大学院大学複合科学研究科情報学専攻助教（兼任）．相互結合網と並列処理に関する研究に従事．

天野 英晴（正会員）

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