非最短経路を用いたチップ内ネットワーク向け経路設定手法

松谷宏紀[†]　鯉渕道紘[†]　山田　裕[†]
上樂明也[†]　天野英晴[†]

本論文では，チップ内ネットワーク向けに *flee* と呼ばれる固定型ルーティング法を提案する．*flee* では非最短経路を導入することで，局所性の強いトラフィックをネットワーク全体に分散させ，スループットを向上させる．近年の Systems-on-a-Chip 設計では，アプリケーションは SystemC に代表されるシステムレベル言語で記述され，設計の初期段階からシミュレーションされる．この段階で各ノードのタスク割り当てが決まるので，ノード間の通信パターンを解析できる．*flee* では，この解析結果をもとに，多量のデータが流れるソース-ディスティネーション・ペアに優先的に最短経路を割り当てる．一方，データ転送量の少ないペアは高負荷なチャネルを避けるように経路が張られるため，非最短経路を取ることがある．実アプリケーションを用いた評価では，*flee* ルーティング法を用いることによって dimension-order ルーティングより最大 22.2%スループットが向上した．

Non-Minimal Routing Strategy for Networks-on-Chips

Hiroki Matsutani,[†] Michihiro Koibuchi,[†] Yutaka Yamada,[†]
Akiya Jouraku[†] and Hideharu Amano[†]

We propose a deterministic routing strategy called *flee* which introduces non-minimal paths in order to distribute traffic with a high degree of communication locality in Networks-on-a-Chip. In the recent design methodology, target system and its application of the Systems-on-a-Chip are designed in system level description language like SystemC, and simulated in the early stage of design. The task distribution is statically decided in this stage, and the amount of traffic between nodes can be analyzed. According to the analysis, a path that transfers a large amount of total data is firstly assigned with a relaxed limitation, thus it is mostly minimal. On the other hand, paths for small amount of total data, are secondly established so as not to disturb previously established paths, thus they are sometimes non-minimal. Simulation results show that the *flee* routing strategy improves up to 22.2% of throughput against the dimension-order routing on typical stream processing application programs.

1. はじめに

Systems-on-a-Chip（SoC）において，IP コア同士を結合するチップ内結合網は，アプリケーションの性能とコストを決定付ける一要素である．チップ内結合網としてオンチップバスが広く用いられているが，バスにつながる全ての IP コアが時分割で 1 つの通信路を共有するため，バスにつながるノード数が増えるとバスが通信のボトルネックとなる．また，近年の集積技術では，ゲート遅延よりも配線遅延の影響が深刻であり，長い配線を構成するバス構造は動作周波数においても不利である．このようなバス構造に起因する問題を解決する次世代チップ内結合網として，チップ内ネットワーク（Networks-on-a-Chip）[1)2)3)] が注目されている．

チップ内ネットワークでは，従来，並列計算機や System Area Network（SAN）で用いられてきたパケット転送技術をチップ内の IP コア間のデータ転送に応用する．データを送信する場合，送信元ノード☆がデータにヘッダを加えパケット化し，中継ノードがこれを転送，最終的に宛先ノードでデータが取り出される．複数のパケットが同時に同一チャネルを取り合わない限り，複数パケットを並列に転送できるので，バスよりも通信帯域に優れる．また，パケットを構成する各フリットは固定長のリンク上を移動し，中継ノードごとにバッファリングされる．そのため，グローバル配線はいくつかの隣接ノード間配線に置き換えられ，配線遅延の問題も解決できる．

† 慶應義塾大学大学院理工学研究科
Graduate School of Science and Technology, Keio University

☆ 本論文では，チップ内ネットワークによって結合される IP コアをノードと呼ぶ．

ネットワーク構造には，本来，様々なアプリケーションに適用するための柔軟性やスケーラビリティが求められる．しかし，SoC では一度製造された後にノード数，ノードの性能や機能，ネットワークトポロジが変化することは考えにくい．さらに，ノードごとに持つルータは小規模なほうが好ましいため，チップ内ネットワークにおいては，汎用性を重視するよりも個々のアプリケーションに特化したアーキテクチャのほうが適している．

SoC の主要なアプリケーションは組込み機器であり，メディア処理や通信分野では高負荷なストリーム処理が行われることも多い．近年の SoC 設計では，対象アプリケーションは SystemC などのシステムレベル言語で記述され，設計の初期段階からシミュレーションされる．この段階で，各ノードのタスク割り当てが決まるのでノード間のデータ転送量を見積もることができる．Ho ら[4] は，見積もった通信パターンを解析し，アプリケーションに適したトポロジを生成する設計手法を提案した．しかし，チップ内ネットワークにおける経路設定では，並列計算機や SAN 向けの通信パターンを考慮しない最短経路ルーティング[5] が用いられているのが現状である．

本論文では，*flee* と呼ばれるチップ内ネットワーク向け固定型ルーティング法を提案する．*flee* では SoC の設計段階で得られるデータ転送量の見積もりをもとに，非最短経路を含めた経路設定を行う．ストリーム処理ではアクセスが一ヶ所に集中するなど，通信パターンに強い局所性が出やすい．この場合，トラフィックが同一チャネルに集中しないように経路を分散させることが望ましいが，最短経路だけでは経路長が短く十分にトラフィックを分散できるとは限らない．そこで，*flee* ルーティング法では，非最短経路を導入することで代替経路の候補数を増やし，高負荷なチャネルを確実に回避する．

本論文では，まず，2 章で SoC の主要なアプリケーションであるストリーム処理の特徴を述べる．3 章でチップ内ネットワークで利用可能な既存のデッドロックフリーなルーティング法を説明し，4 章で *flee* ルーティング法を提案する．そして，5 章で実アプリケーションを用いた *flee* の評価結果を示し，6 章で本論文をまとめる．

2. ストリーム処理

Viterbi デコーダや JPEG，MPEG コーデックのようなストリーム処理では，ひとかたまりのデータに対し一連の処理が実行される．本論文では，このような一連の処理の流れのうち 1 つの処理単位をタスクと呼ぶ．**図 1** に JPEG2000 デコーダのタスクの流れを示す．各タスクはそれぞれ各ノードに割り当てられ，パイプライン的に動作する．図 1 ではタスク間通信は隣接ノード間に限られている．この JPEG2000 の例では，EBCOT（Embedded Block Coding with Optimal Truncation）の計算負荷が高く，これらをそれぞれ 1 つの計算ノードで実行すると EBCOT が処理のボトルネックとなる．この場合，**図 2** に示すように，EBCOT を複数のノードに分散し並列に実行することで，ストリーム処理の負荷を均等にできる．図 2 の並列化されたモデルでは，ノード間通信に分散と収集が発生する．

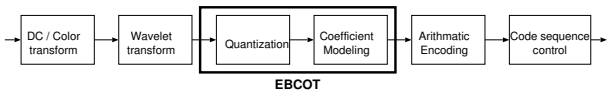

図 1 JPEG2000 デコーダのタスクの流れ．EBCOT の負荷が高い．

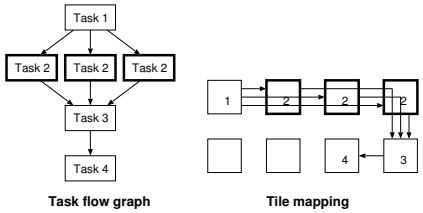

図 2 タスクの並列化．高負荷なタスク 2 を 3 つのノードに分散，並列処理する．

チップ内ネットワークでは，2 次元メッシュやトーラスなど単純なネットワークトポロジが利用されることが多い．タスクをこのようなトポロジに割り当てると，図 2 に示すようにノード間の通信に高い局所性が生じる．このような hot-spot はパケットの衝突を引き起こし，スループット低下の原因となる．したがって，高い局所性が生じる箇所を避けるかたちでトラフィックを分散させれば性能を改善できる．

3. 既存のルーティング法

既存のチップ内ネットワークではデッドロックフリーな固定型ルーティングが多用される．適応型ルーティングではパケットごとに動的に通信経路が選択されるが，固定型ルーティングでは静的に経路が定まる．固定型ルーティングの利点は，出力チャネルを動的に選択する機能が不要なことと，パケット転送の FIFO 性が保証されることである．

チップ内ネットワークで広範囲に利用されている固定型ルーティングとして，dimension-order ルーティング[5] が挙げられる．dimension-order ルーティングでは，2 次元メッシュやトーラスにおいて，まず，送信元ノードから x 軸方向のチャネルを使って移動し，次に，y 軸方向のチャネルを使って宛先ノードに到達する．

一方，適応型ルーティングによって提供された経路集

合の中から 1 つの経路をあらかじめ選択し，静的に経路を設定することで固定型ルーティングを実現できる．しかし，既存の経路選択アルゴリズム[6] は様々な並列プログラムが実行される並列計算機や PC クラスタ向けに設計されているため，データ転送量など通信パターンに応じて経路を選択する機能がない．

図 2 で示したように分散と収集を含む通信パターンでは，ソース-ディスティネーション・ペアごとにデータ転送量が大きく異なる．また，パイプライン処理のメインストリーム同士が同一チャネルを取り合うようでは性能は出ない．よって，通信パターンに応じて経路を構築する手法を提案することで，既存のルーティング技術に比べチップ内ネットワークのスループットを向上できると考えられる．

4. *flee* ルーティング法

本章では，非最短経路を活用する固定型ルーティングとして *flee* ルーティング法を提案する．2 章で述べたとおり，ストリーム処理の通信パターンには強い局所性が生じやすい．そこで，*flee* ルーティング法では，非最短経路を導入することで代替経路集合の数を増やし，経路を分散させて混雑を緩和する．その際，各ソース-ディスティネーション・ペアのデータ転送量を考慮して経路を設定する．具体的には，多量のデータが流れるソース-ディスティネーション・ペアには優先的に最短経路を割り当て，データ転送量の少ないペアは高負荷なチャネルを避けるような経路を設定する．

flee ルーティング法は次の 2 ステップから構成される．(1) 通信パターンの静的な解析，(2) デッドロックフリーな条件のもとでの経路設定．

4.1 通信パターンの解析

近年の SoC 設計では，対象アプリケーションは SystemC や SpecC などのシステムレベル言語で記述され，設計の初期段階からシミュレーションされる．この段階でノードへのタスク割り当てが決まり，SystemC レベルのシミュレーションによって通信パターンを解析できる．この通信パターンの解析結果をもとに，以下の手順で各ソース-ディスティネーション・ペアに優先順位を付ける．

(1) ソース-ディスティネーション・ペアごとに，対象アプリケーションで発生するデータ転送量の合計を求める．
(2) データ転送量の合計値が多い順に，ソース-ディスティネーション・ペアを並び換える．

図 3 を用いて通信パターンの解析手順を説明する．まず，SoC 設計のシミュレーション結果をもとに，パケット発生時のクロック，送信元ノード，宛先ノード，データサイズから成る通信パターンを得る．そして，データ転送量の多いソース-ディスティネーション・ペアから順に高い優先順位を与え，これを “analysis record” として記録する．次のステップでは，データ転送量が多く高い優先順位を得たソース-ディスティネーション・ペアから優先的に最短経路が割り当てられる．

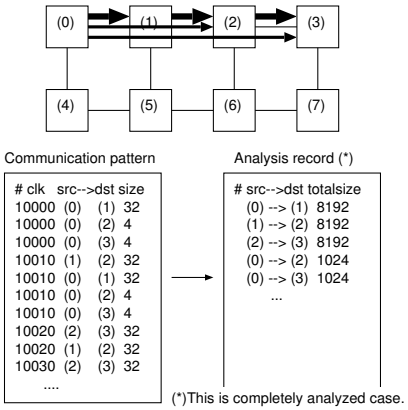

図 3 通信パターンの解析．データ転送量の多い順にソース-ディスティネーション・ペアをソートする．

ルーティング段階に及んでも，アプリケーションのデータ転送量が見積もれない状況がまれに起こり得る．つまり，図 3 の通信パターンで送信元ノードと宛先ノードのみが限定されている場合である．データ転送量を含んだ analysis record のほうが経路を分散させるためには有利であるが，*flee* はデータ転送量が不明な場合にも適用できる．このような不完全な analysis record と完全な analysis record を用いた *flee* の性能比較は 5.2.2 節で示す．

4.2 経路設定

前節で説明した analysis record の優先順位をもとに，各ソース-ディスティネーション・ペアにデッドロックフリーな経路を割り当てる．ここで，ネットワーク内の全てのチャネルにコストという指標を導入する．代替経路が複数ある場合，それぞれの経路で利用されるチャネルのコストが比較され，コストが最も小さい経路が採用される．経路設定手順を次に示す．

(1) 全てのチャネルのコストを 1（minimum channel cost）に初期化する．
(2) 経路が設定されていないソース-ディスティネーション・ペアの中から一番優先順位が高いものを 1 つ選ぶ．そのペアに対して：
  (a) Dijkstra 法☆を用いて，デッドロックフリーの条件を満たす最小コストの経路（必ずしも最短経路とは限らない）を選択する．

☆ 重み付き有向グラフにおいて最短経路を発見するアルゴリズム．

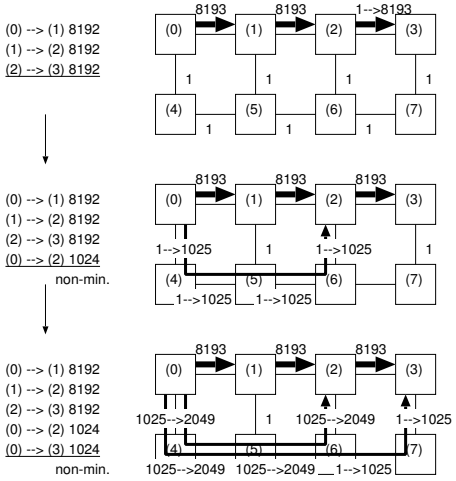

図 4 *flee* における経路設定．(0)-(2) 間と (0)-(3) 間の通信に非最短経路が割り当てられている．

(b) 選択した経路が通過する全てのチャネルのコストを増加させる．増分は，そのペアによるデータ転送量の合計値，または，1 とする．

(3) すべてのソース-ディスティネーション・ペアの経路が定まるまでステップ (2)-(3) を繰り返す．

ステップ (2)(a) のデッドロックフリー保証は，Turn-Model[7] など適応型ルーティング等で用いられるデッドロックフリー条件を課すことで実現する．ステップ (2)(b) でソース-ディスティネーション・ペアが完全に解析できた（データ転送量が判明している）場合，各チャネルコストにデータ転送量の合計値を加算する．データ転送量の合計値の単位はビット長またはバイト長などである．一方，完全に解析できなかった場合，データ転送量が不明なので単純に 1 を加算する．

図 4 に *flee* による経路設定の例を示す．この例は，ソース-ディスティネーション・ペアのデータ転送量が完全に解析できた場合を想定しており，デッドロックフリー条件として West-first Turn-Model[7] を採用した．図 4 に示すように，優先順位の高いソース-ディスティネーション・ペアは最初に経路が設定されるため，結果的に最短経路が割り当てられる．経路設定が進むにつれて各チャネルコストは初期値の 1 から増加し，hot-spot となるチャネルではコストが高くなる．優先順位の低いペアでは，hot-spot を避けるように経路が張られるため，非最短経路を取ることがある．このように *flee* ルーティング法では，通信のメインストリームに最短経路を割り当てる一方，優先順位の低い通信には，ネットワークリソースを有効活用するために非最短経路を割り当てることがある．

flee ルーティング法の計算量は $O(n^2k)$ である．ただし，n はネットワーク中のノード数，k は経路を設定するソース-ディスティネーション・ペアの数とする．

5. 性 能 評 価

多くのストリームアプリケーションでは設計時に通信量を解析できる．そのため，まず，完全に解析された analysis record を用いて 2 次元メッシュおよびトーラス上で *flee* ルーティング法を dimension-order ルーティングと比較する．次に，完全な analysis record と不完全な analysis record を用いた *flee* の性能比較を行う．

5.1 シミュレーション環境

5.1.1 ネットワークモデル

評価のために C++言語で記述されたフリットレベル・シミュレータを実装した．評価対象のトポロジとして，チップ内ネットワークで一般的な 2 次元メッシュとトーラス[1)2)3)] を用いる．各ルータのスイッチング機構には，チャネルバッファ，クロスバ，リンクコントローラ，コントロール回路を単純化したモデルを採用した．ヘッダフリットの転送には 3 サイクルかかる．具体的には，ルーティングに 1 サイクル，フリットがクロスバを通り入力チャネルから出力チャネルに到達するのに 1 サイクル，フリットが次のルータまたはホストに到達するのに 1 サイクルである．シミュレーション時間は 1,000,000 サイクル以上とした．

5.1.2 トラフィックパターン

flee ルーティング法では通信パターンをもとに経路を設定するため，実アプリケーションの通信パターンを用いて評価することが望ましい．そこで，典型的なストリーム処理アプリケーションとして JPEG コーデック，Viterbi デコーダのトレースデータを評価に用いる．これらの実装では，ストリーム処理の各タスクは最大 16 個のノードに割り当てられた．アプリケーションは C 言語ベースで開発され，データ転送量も C 言語レベルのシミュレーションによって解析された．*flee* による経路設定では，ソース-ディスティネーション・ペアに経路が設定される度に，その経路で利用されるチャネルのコストにデータ転送量が加算される．本論文ではデータ転送量の単位としてビット長を用いた．

比較のため，実アプリケーションのトレースデータに加え，ユニフォームトラフィックも評価に用いる．このユニフォームトラフィックでは，全てのノードが自分以外のノードにランダムにパケットを送信する．パケット長は 259 フリットとし，そのうち 2 フリットはヘッダとした．

5.2 シミュレーション結果

5.2.1 完全な静的解析を用いた場合

2 種類のストリームアプリケーションとユニフォームトラフィックを用いて，*flee* ルーティング法と dimension-order ルーティングを比較する．本論文では，*flee* ルー

ティング法のデッドロックフリー制約条件として West-First Turn-Model[7] を適用した．この場合，*flee* および dimension-order ルーティングが利用する仮想チャネル数はメッシュで 1，トーラスで 2 である．

図 5 と**図 6** は，Viterbi トレースを用いた際のスループット（accepted traffic）とレイテンシを表しており，前者はメッシュ，後者はトーラスでの評価結果である．グラフ中の “Flee” は *flee*，“DOR” は dimension-order ルーティングを表し，それぞれの平均ホップ数を括弧内に示した．2 次元メッシュトポロジ（図 5）では，*flee* の平均ホップ数は 2.52，dimension-order ルーティングは 1.84 となり，*flee* が実際に非最短経路を取っていることが確認できる．このように非最短経路を導入することで，*flee* ルーティング法は経路を分散させ，dimension-order ルーティングよりスループットを 14.2%向上できた．一方，2 次元トーラストポロジ（図 6）では，*flee* は dimension-order ルーティングよりスループットを 22.2%向上させることができた．2 次元トーラスではラップアラウンドチャネルが利用できるため，*flee* はこの分代替経路の候補数を増やすことができ，メッシュのときよりも有利となった．

図 7 と**図 8** は，それぞれメッシュとトーラスにおける JPEG トレースを用いた際のスループットとレイテンシのグラフである．この JPEG の実装では，メインストリームのデータ処理は逐次的に実行される．さらに，逐次的に実行される各タスクは，平均ホップ数が最小になるように配置されているため，ほとんどのノード間通信は隣接ノード間だけで完結している．そのため，*flee* で非最短経路はほとんど利用されず，dimension-order ルーティングとの性能差もほとんど生じなかった．

本来，*flee* ルーティング法は高い局所性を持ったトラフィックパターンに対処するために設計されている．ここではワーストケースでの *flee* の振る舞いを調べるため，局所性のないユニフォームトラフィック上で評価を行った．**図 9** は，2 次元メッシュにおけるユニフォームトラフィックを用いた際のスループットとレイテンシのグラフである．ユニフォームトラフィックでは dimension-order ルーティングによって十分に経路を分散させることができており，非最短経路を導入してもこれ以上の経路分散は望めない．そのため，非最短経路の導入はネットワークリソースを余計に消費するだけで hot-spot を緩和できず，結果のグラフでも *flee* の性能は dimension-order ルーティングに劣っている．グラフ中の平均ホップ数を見ると，*flee* ルーティング法はユニフォームトラフィックにおいてもいくつか非最短経路を取っていることがわかる．これは，analysis record に記された優先度順に，すでに設定した経路と重複しないように経路を張るからである．

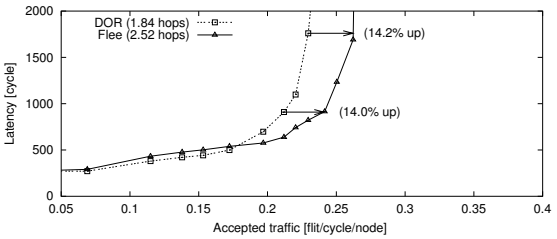

図 5 Viterbi トレース（4 × 4 メッシュ）．

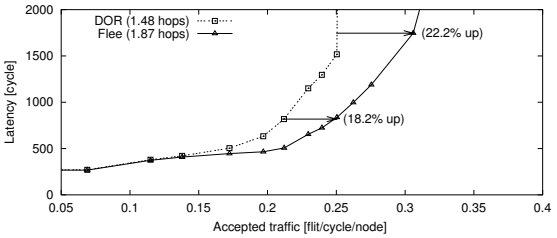

図 6 Viterbi トレース（4 × 4 トーラス）．

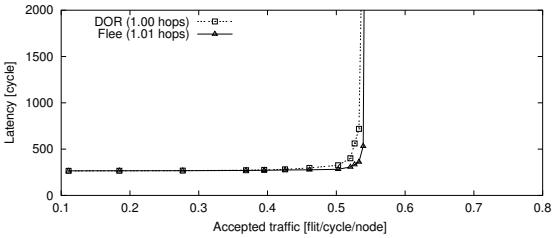

図 7 JPEG トレース（4 × 4 メッシュ）．

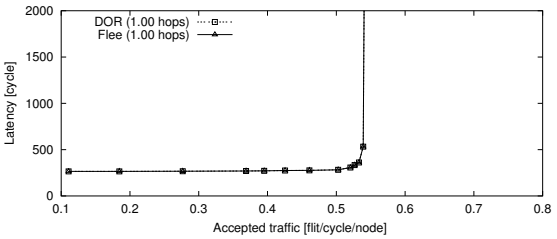

図 8 JPEG トレース（4 × 4 トーラス）．

ストリーム処理ではトラフィックに高い局所性を含むことが多い．このような通信パターンにおいて，*flee* ルーティング法は経路を分散させ，性能を向上させることができた．また，メッシュとトーラスでの性能比較より，利用可能なリンク数が多いほうが，*flee* の dimension-order ルーティングに対する優位性が高まることがわかった．

5.2.2 不完全な静的解析を用いた場合

flee ルーティング法における analysis record の影響を確認するため，完全な analysis record と，データ転送量が判明していない analysis record で性能比較を行った．ここでは Viterbi トレースを用いて 2 次元メッシュとトーラス上で測定した結果（**図 10** と**図 11**）を示す．グラフ中の “Flee.in” は不完全な analysis record を用いた場合，“Flee” は完全な analysis record を用いた場合を表している．不完全な analysis record を用いた *flee* の性能は，

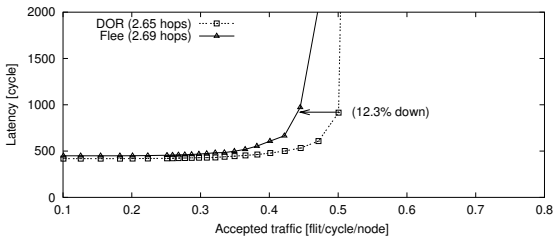

図 9 ユニフォームトラフィック（4 × 4 メッシュ）.

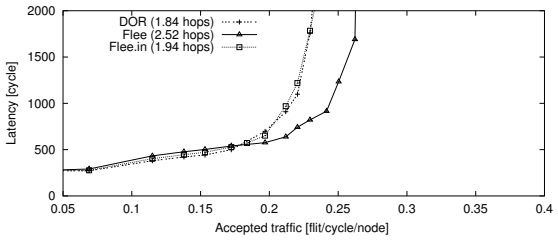

図 10 不完全な静的解析を用いた Viterbi トレース（4 × 4 メッシュ）.

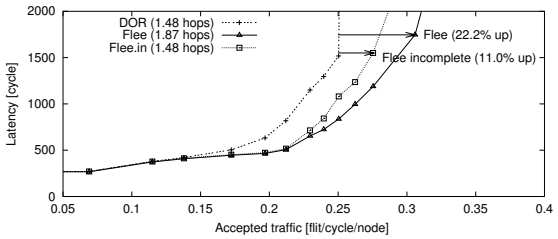

図 11 不完全な静的解析を用いた Viterbi トレース（4 × 4 トーラス）.

図 11 では dimension-order ルーティングより 11.0%スループットを向上できたが，図 10 では dimension-order ルーティングと同程度の性能しか出ていない．このように不完全な analysis record を用いると *flee* の性能は安定しない．これは，各ソース-ディスティネーション・ペアのデータ転送量が経路を最適化する上で重要な要素であることを示している．4 章で述べたとおり，多くのストリーム処理アプリケーションでは設計時にデータ転送量を見積もることができる．しかしながら，ルーティング段階に及んでもデータ転送量が判明しない場合がわずかながら想定され，その場合に用いられる不完全な analysis record では，完全な analysis record ほど安定して性能を向上できないことがわかった．

6. ま と め

本論文では，チップ内ネットワーク向けに *flee* と呼ばれるデッドロックフリーな固定型ルーティング法を提案した．チップ内ネットワークでは SoC の設計段階で通信パターンを予測できる．これらの通信パターンには強い局所性が出やすいが，最短経路だけでは経路長が短く十分にトラフィックを分散できるとは限らない．そこで，*flee* ルーティング法では，非最短経路を導入することで代替経路の候補数を増やし，高負荷なチャネルを確実に回避する．*flee* では SoC 設計の初期段階から得られるデータ転送量の見積もりをもとに，多量のデータが流れるソース-ディスティネーション・ペアに優先的に最短経路を割り当てる．一方，データ転送量の少ないペアは高負荷なチャネルを避けるように経路が張られるため，非最短経路を取ることがある．実際のストリームアプリケーションを用いたシミュレーションでは，*flee* ルーティング法は非最短経路を用いることで経路を分散でき，既存の最短経路ルーティング法より最大 22.2%スループットを向上できた．*flee* ルーティング法はトラフィックパターンが完全に解析できた場合に加え，データ転送量が不明な場合にも適用できる．両者の性能比較より，データ転送量を考慮した analysis record を利用することができれば安定してスループットを向上できることがわかった．

参 考 文 献

1) W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the 38th Design Automation Conference*, pp. 684–689, June 2001.
2) T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins. Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs. In *Proceedings of the Field-Programmable Logic and Applications (FPL)*, pp. 795–805, September 2002.
3) J. Liang, A. Laffely, S. Srinivasan, and R. Tessier. An Architecture and Compiler for Scalable On-Chip Communication. *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 12, No. 7, pp. 711–726, July 2004.
4) Wai Hong Ho and T. M. Pinkston. A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns. In *Proceedings of the Ninth International Symposium on High-Performance Computer Architecture*, pp. 377–388, February 2003.
5) W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transaction on Computers*, Vol. 36, No. 5, pp. 547–553, May 1987.
6) J. C. Sancho and A. Robles. Improving the Up*/Down* Routing Scheme for Networks of Workstations. In *Proceedings of the European Conference on Parallel Computing*, pp. 882–889, August 2000.
7) C. J. Glass and L. M. Ni. The Turn Model for Adaptive Routing. *Proceedings of International Symposium on Computer Architecture*, pp. 278–287, 1992.