メニーコアにおけるタスクの移動を支援する On-Chip Micro Mobility プロトコルの設計

松谷宏紀[†1]　鯉渕道紘[†2]　天野英晴[†1]

メニーコアチップにおいて，一部のコアへの負荷の集中は性能のボトルネックを生み，局所的な温度上昇，および，それに伴うリーク電力の急増を招く．本論文では，多数のコアを利用できるというメニーコアの利点を活かし，あるコアで動作していたタスクを別のコアへ移動させることで，このような問題の解決を試みる．このとき問題となるのは，タスクの移動に伴う物理的なアドレスの変化である．つまり，現在このタスクと通信しているすべての通信相手に，移動先のアドレスを通知しなければ，タスクの移動後に計算を継続することができなくなる．本論文では，メニーコアにおけるタスクの移動透過性，および，着信可能性を保証するための通信プロトコルとして On-Chip Micro Mobility を Network-on-Chip 上に設計したので報告する．

Design of the On-Chip Micro Mobility Protocol Supporting Task Migration for Many-Core Chips

Hiroki Matsutani,[†1] Michihiro Koibuchi[†2]
and Hideharu Amano[†1]

In a many-core chip, concentration of computational load on a specific core introduces a serious performance bottleneck, a locally-increased temperature, and a rapidly-increased leakage current. One of the attractive solutions to these problems is a task migration from heavily-loaded core to a free core, since many processing cores are available in such chips. Since the physical location and address of the core is changed after the migration, its new address must be notified to all corresponding tasks that will interact with the core in order to continue their communication. Here we have designed the communication protocol that guarantees the transparency and reachability over the task migrations for many-core chips. This paper illustrates the design of the On-Chip Micro Mobility protocol.

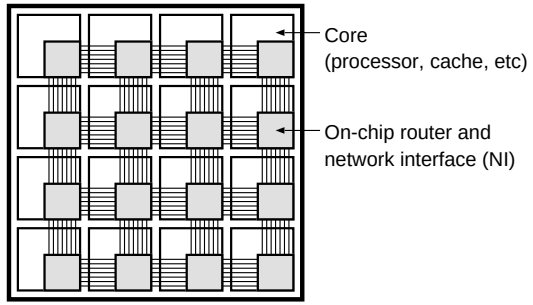


図 1　Network-on-Chip（NoC）の例．この例ではチップは 16 個のタイルに分割され，各タイルはコアおよびルータ回路を持つ．コアはプロセッサなどの演算主体であり，ルータはコア間のパケット転送を担う．ルータ同士はオンチップリンクを介して相互接続される．コアとルータの間にはネットワークインタフェイス（NI）が実装される．

1. はじめに

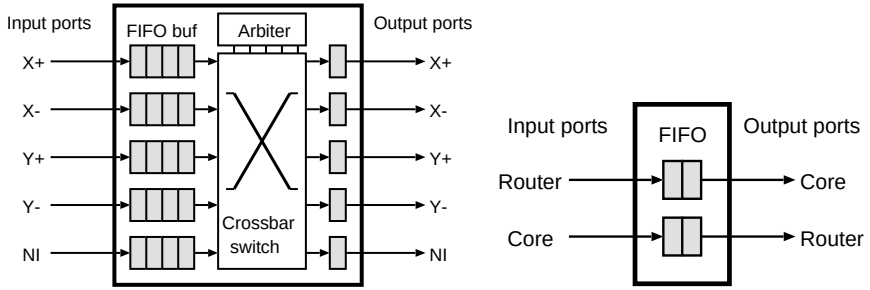
1.1 Network-on-Chip について

半導体技術の進歩によって，1 チップ上に実装できるプロセッサコアの数が増加の一途を辿っている．近年では，64 個以上のプロセッサコアを内蔵したプロトタイプチップ，および，商用チップがいくつも報告されている[1)]．このようなたくさんのコアを接続するために，古典的なオンチップバスに代わり，**図 1** に示す Network-on-Chip（NoC）[2),3)] がメニーコアチップ向けの結合網として広く用いられるようになった．

NoC では，コアから出力されたデータはネットワークインタフェイス（NI）でパケット化され，複数のルータを経由して宛先ルータまで転送される．宛先ルータまで到達したパケットは NI でヘッダを取り除かれ，宛先コアへと渡される．**図 2** にオンチップルータと NI の構造を示す．また，このようなルータ間でやり取りされるパケットのフォーマットの例を**図 3** に示す．

オンチップルータによる単純なパケット転送制御に加え，さらに，その上位レイヤとして，NoC においても高機能な通信プロトコルが実装されるようになった．NoC における通信プロトコルの例として，end-to-end のフロー制御[3)]，パケットのエラー検出・再送，エラー訂正[5)]，フラグメンテーション，キャッシュのコヒーレンス制御などがある．本論文では，この中でメニーコアにおけるタスクマイグレーションを可能とする通信プロトコルを提案，設計する．

†1 慶應義塾大学大学院 理工学研究科
Graduate School of Science and Technology, Keio University
†2 国立情報学研究所 / 総合研究大学院大学
National Institute of Informatics / The Graduate University for Advanced Studies



(a) オンチップルータ回路.　(b) ネットワークインタフェイス（NI）回路.

図 2 NoC のハードウェア．このルータは 5 入力 5 出力の wormhole ルータ[4] である．入力されたパケットは 1) 入力バッファで蓄えられ，2) パケットヘッダに格納された宛先アドレスから転送に使用する出力ポートを計算する．そして，3) パケットを出力するためにクロスバスイッチのアービトレーションを行い，4) 出力ポートの利用権を得た後，パケットはクロスバ上を通過して出力ポートから出力される．

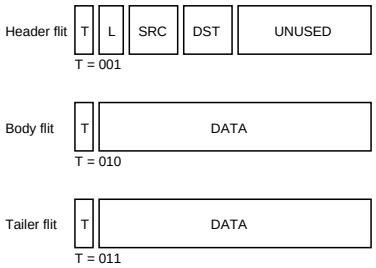
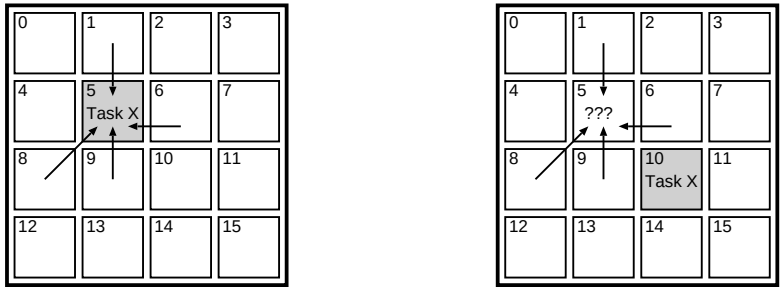


図 3 通常のパケットフォーマット．パケットは 1 個のヘッダフリット，0 個以上のデータフリット，1 個のテイラフリットから成る．Type（T）フィールドはフリットのタイプを表す．Length（L）フィールドはパケットに含まれるフリット数を示す．SRC および DST フィールドは送信元アドレスと宛先アドレスである．

1.2 タスクマイグレーションについて

メニーコアチップにおいて，一部のコアへの負荷の集中は性能のボトルネックを生み，局所的な温度上昇，および，それに伴うリーク電力の急増を招く．多数のコアが利用できるメニーコアチップでは，負荷が集中しているコアで動作しているタスクを別のコアへ移動させることでこのような問題を解決できると我々は考えている．例えば **図 4** のように，コア 5 で動作している Task X をコア 10 へ移すことで，コア 5 へのアクセスの集中，および，局所的な温度の上昇を緩和できると考えられる．

このとき問題となるのは，タスクの移動に伴う物理的なアドレスの変化である．つまり，現



(a) Task X の移動前.　(b) Task X の移動後.

図 4 タスクの移動に伴う問題．この例ではコア 5 の上で Task X が動作しており，コア 1，6，8，9 上のタスクと通信をしている．その後，Task X はコア 10 へ移動する．Task X の移動後，それまで Task X と通信していたコア 1，6，8，9 上のタスクたちは通信を継続できなくなる．

在このタスクと通信しているすべての通信相手に移動先のアドレスを通知しなければ，タスクの移動後に計算を継続することができなくなる．図 4 の例では，Task X の移動後，これまでの通信を継続できなくなっている．ここでは，タスクの移動によって通信を維持できることを**移動透過性**と呼び，タスクの物理的な位置が変化しても同じアドレスでアクセスできることを**着信可能性**と呼ぶ．本論文では，メニーコアにおけるタスクの移動透過性，および，着信可能性を保証するための通信プロトコルを NoC 上に設計したので報告する．

1.3 本論文の構成

本論文では，NoC におけるタスクの移動透過性と着信可能性を保証するために Mobile IP を参考にした．まず 2 章で Mobile IP について説明する．そのうえで，3 章で On-Chip Micro Mobility プロトコルを提案し，4 章では，これを近接ノード単位の移動に拡張した On-Chip Network Mobility を提案する．これらの NoC への実装について 5 章で検討し，簡単な予備評価を 6 章で示す．最後に 7 章で本論文のまとめと今後の課題を述べる．

2. Mobile IP

Mobile IP は，Internet Protocol（IP）を用いた通信において，移動透過性および着信可能性を保証するための通信プロトコルである[6),7)]．Mobile IP はノード単位の移動を想定した通信プロトコルであるが，これをネットワーク単位の移動に拡張した Network Mobility も標準化がされている[8)]．本論文で提案する On-Chip Micro Mobility および On-Chip Network Mobility は Mobile IP と Network Mobility の概念を利用している．以下ではその概要を紹介する．

インターネットで用いられる IP アドレスには，ネットワークの識別子とそのネットワーク内での識別子の両方の意味がある．例えば，あるノードが Network A から Network B に移動すると，そのノードのアドレスは変化する．これにより，ネットワークの移動後，このノードはそれまでの通信を継続できなくなる．さらに，これまでの通信相手は，移動ノードに対し，移動前と同じ識別子を用いてアクセスする（相手を見付ける）ことができなくなる．

Mobile IP では，移動ノードに対し，どのネットワークに接続していても変化しない固有のアドレス（Home Address，HoA）を付与する．そして，移動ノードが訪問先で得る実際のアドレス（Care-of Address，CoA）と HoA の対応付け（binding）を Home Agent と呼ばれるサーバが管理する．こうすることで，移動ノードの HoA 宛てのパケットを現在の CoA 宛てに転送できるようになる．つまり，移動ノードがどのネットワークに接続していても，HoA を用いてこの移動ノードにアクセスできる．

以降では，この考え方をメニーコアにおけるタスクの移動透過性，および，着信可能性の保証のために用いる．

3. On-Chip Micro Mobility

本章では On-Chip Micro Mobility プロトコルを提案する．On-Chip Micro Mobility によって，Mobile Task と呼ばれるタスクがどのコア上で動作していたとしても，Mobile Task ごとに固有の識別子でアクセスできる．つまり，この識別子宛てにパケットを送信すれば，Mobile Task がどのコア上で動作していたとしても必ずこのタスクにパケットが届く．

まず，3.1 節で用語を定義し，その後の節でプロトコルの動作を説明する．

3.1 用語の定義

本論文ではコアの上で動作するプロセスをタスクと呼ぶ．タスクには以下の 2 種類がある．

- Mobile Task（MT）：動作中に別のコアに移動できるタスク．
- Corresponding Task（CT）：MT と通信するタスク．

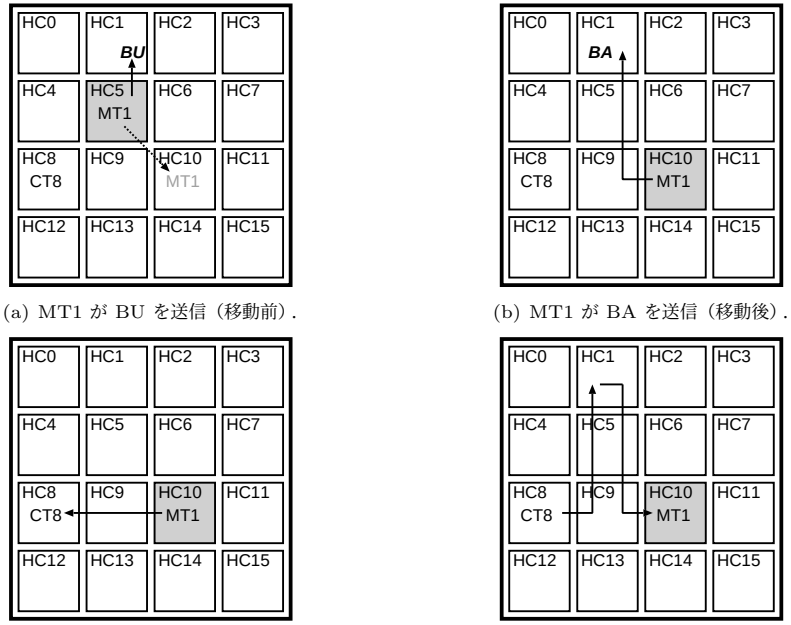
MT 同士が通信する場合，両者に MT と CT の両方の機能が必要となる．とくに説明がない限り，すべてのタスクは MT の CT の両方の機能を持つものとする．

MT から見て，コアには以下の 2 種類がある．

- Home Core（HC）：MT ごとに必ず 1 個のコアを HC としなければならない．HC は MT が別のコアで動作している間，MT のプロキシとして動作する．
- Foreign Core（FC）：HC 以外のコアをすべて FC と呼ぶ．

MT がパケット通信に用いるアドレスには以下の 2 種類がある．

- Home Address（HoA）：MT が HC で動作しているときのアドレス．通信相手である CT は On-Chip Micro Mobility プロトコルを用いることで，MT がどのコアに移動していても常に HoA を用いて MT にアクセスできる．
- Care-of Address（CoA）：MT が HC の外部のコア（つまり FC）に移動したときに得られる実際のネットワークアドレス．



(a) MT1 が BU を送信（移動前）．
(b) MT1 が BA を送信（移動後）．
(c) MT1 から CT8 への通信．
(d) CT8 から MT1 への通信．

図 5 On-Chip Micro Mobility による通信（route optimization 無し）．MT1 はコア 5 からコア 10 へ移動している（図 5(a)-図 5(b)）．CT8 から MT1 への通信は Home Core を経由する必要がある（図 5(d)）．

以降，MT が HC 上で動作するとき（3.2 節），および，MT が FC に移動するとき（3.3 節と 3.4 節）について On-Chip Micro Mobility プロトコルの挙動を説明する．

3.2 Mobile Task が Home Core にいるとき

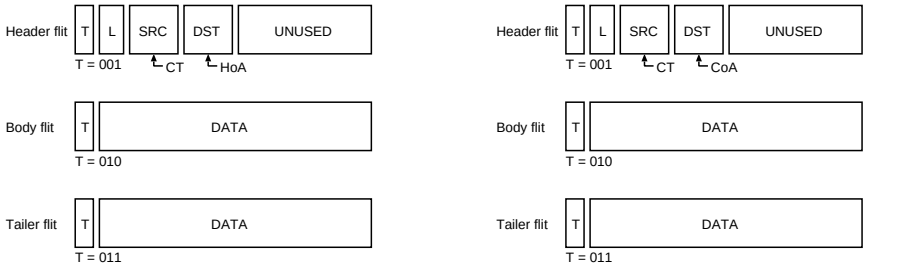
MT が HC 上で動作するとき，MT の CoA は HoA と等価であり，On-Chip Micro Mobility プロトコルを用いなくとも HoA を用いて MT にアクセスできる．

Mobile Task から Corresponding Task への通信

パケットフォーマットは通常の通信（図 3）と同じである．このとき SRC は HoA（CoA と等価）とし，DST は CT のアドレスとなる．

(a) Binding Update（BU）パケット. (b) Binding Acknowledge（BA）パケット.

図 6 HoA と CoA の binding を管理するための制御パケットフォーマット．BU は MT が移動前に新しい CoA を HC に通知するためのメッセージであり，BA は MT の移動が完了したことを HC に通知する．



(a) パケットフォーマット（CT から HC）. (b) パケットフォーマット（HC から MT）.

図 7 On-Chip Micro Mobility において，route optimization しないときの CT から MT へのパケットフォーマット．パケットは HC で中継される．CT から HC までの転送（図 7(a)）では MT の HoA が DST であり，HC から MT までの転送（図 7(b)）では MT の CoA が DST となる．

Corresponding Task から Mobile Task への通信

パケットフォーマットは通常の通信（図 3）と同じである．このとき，SRC は CT のアドレスとし，DST は HoA（CoA と等価）となる．

3.3 Mobile Task が Foreign Core に移るとき（Route Optimization 無し）

MT が HC の外部（つまり FC）に移動するとき，MT は訪問先で CoA を得る．この CoA は MT 固有の HoA とは異なるが，On-Chip Micro Mobility プロトコルを用いれば，MT がどのコアに移動しても MT 固有の HoA を用いて MT にアクセスできる．

まず，**図 5** を用いて，route optimization を行わないシンプルな MT の移動を説明する．route optimization については後述する．

Mobile Task による移動の通知

MT は移動前に，以下の手順で，移動先で得る CoA を HC に通知する．

(1) MT の移動先，つまり，移動先の CoA が決まる．図 5(a) の例では，コア 5 で動作している MT1 がコア 10 へ移動しようとしている（新しい CoA は 10 である）．
(2) MT は移動する前に自身の HC に対して Binding Update（BU）を送る．BU メッセージには MT の新しい CoA が含まれているので，HC は MT の HoA と移動先の CoA の binding を得ることができる．図 5(a) の例では MT1 の HC はコア 1 である．そのため MT1 は移動前に HC1 に対して BU を送信している．
(3) MT の新しいコアへの移動が完了した後，MT は HC に対して Binding Acknowledge（BA）を送る（図 5(b)）．

BU および BA メッセージのパケットフォーマットを**図 6(a)** および **図 6(b)** に示す．

MT が HC に対して BU を送信し，移動が完了して再び HC に対して BA を送信するまでの間，MT はパケットを受信することができない．この間，MT 宛てのパケットはすべて HC に届くので（理由は後述），HC は MT の移動が完了するまでの間，1) MT 宛てのパケットを破棄して NACK を送るか，2) MT の移動が完了するまでバッファリングする必要がある．

Home Core における binding の管理

HC は BU を受信したら HoA と CoA の対応付けを Binding Cache（BC）に保存する．BC は **図 8(a)** のような構造を持ち，HoA をキーに MT の現在の CoA を得ることができる．

Mobile Task から Corresponding Task への通信

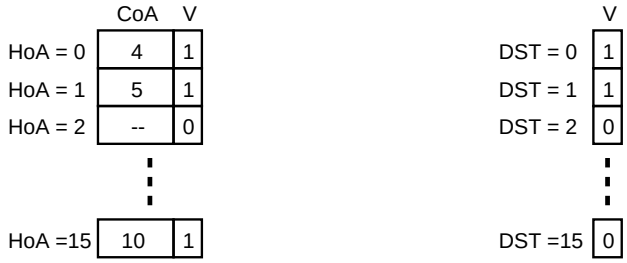
MT から CT への通信では，パケットの SRC を現在の CoA ではなく HoA とする．それ以外は通常の通信と同じである．**図 5(c)** の例では，SRC は 1，DST は 8 となる．

Corresponding Task から Mobile Task への通信

ここでは，MT が CT に対して BU を送信しない場合，つまり，CT は MT の現在の CoA を知らない場合を考える．このとき，CT から MT への通信は以下の手順で行われる．

(1) CT において，MT の HoA 宛てにパケットを生成する．
(2) CT において，BC をひいて，HoA から対応する CoA を検索する．
(3) ここでは CT の BC に MT の binding が登録されていない場合を考えているので，HoA に対応する CoA が見付からない．この場合，CT は HoA 宛てにパケットを送信する（図 7(a)）．
(4) MT の HoA 宛てのパケットが MT の HC に届く．
(5) HC において，BC をひいて，HoA から対応する CoA を検索する．
(6) HC は HoA 宛てのパケットの宛先を対応する CoA に書き換え，パケットを CoA 宛てに転送する（図 7(b)）．なお，転送時には，デッドロックを防ぐために，これまでと異なる仮想チャネルを使用する必要がある（詳細は 5.3 節で述べる）．
(7) パケットが MT の CoA に届く．

このように CT の BC に宛先 MT の binding が登録されていないときは，常に HC を経由した通信となる．図 5(d) の例では，CT8 から MT1 への通信は，MT1 の HoA であるコア 1 に送られ，HC1 によって現在の CoA であるコア 10 へ転送されている．このように HC を経由する通信を「route optimization されていない通信」と呼ぶ．



(a) Binding Cache（BC）の構造.　(b) Binding Update List（BUL）の構造.

図 8 HoA と CoA の binding を管理するためのデータ構造．BC は CT および HC 側の機能であり，通信相手である MT の HoA と CoA の binding を管理する．通信相手が BC に登録されていれば，通信相手の CoA 宛てにパケットを送り，登録されていなければ HoA 宛てに送る．HoA 宛てに送ったパケットは HC にて中継される．BC の Valid（V）フィールドは対応する CoA が有効かどうかを表す．BUL は MT 側の機能であり，通信相手が自分の binding を持っているかどうかを記録する．MT は，移動前に，BUL の V フィールドが 1 になっている CT に対し binding を無効化するための BU を送信しなければならない．

route optimization されない場合，MT 宛てのすべてのパケットが HC を経由することになるため，非最短経路が増え，通信遅延やスループットに影響が生じる．

3.4 Mobile Task が Foreign Core に移るとき（Route Optimization 有り）

次に CT と MT 間の通信において route optimization を行う場合を考える．

Corresponding Task から Mobile Task への通信（Route Optimization 前）

route optimization する場合，MT は HC に加え，route optimization されていないパケットを送信してきた CT に対しても BU を送る．BU を送信した CT を識別するために Binding Update List（BUL）を持つ（BUL の構造は **図 8(b)** を参照）．こうすることで，MT とこの MT から BU を受信した CT との通信は，以後，route optimization される．

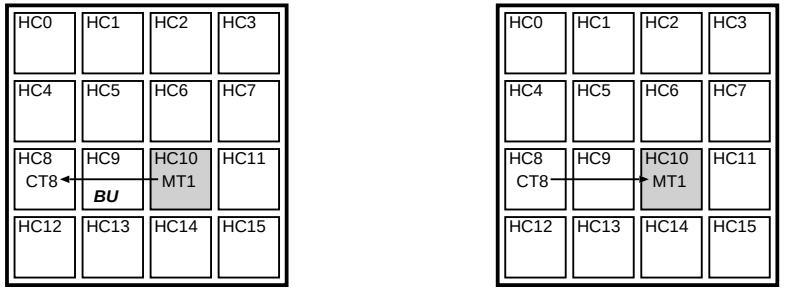
このために，MT では以下の 2 つの処理が必要となる．

(1) MT は CT からパケットを受信する度に BUL を確認し，まだ BU を送っていなければ，その CT に対し BU を送る．
(2) MT は BU を送信する度に BUL を更新する．

図 5(d) の例では，MT1 は CT8 から route optimization されていないパケットを受信した．そのため **図 9(a)** では，MT1 は CT8 に対して BU を送信している．

Mobile Task による移動の通知

基本的に 3.3 節と同じであるが，route optimazation を行う場合は CT が MT の binding を持っている可能性がある．このような binding はパケットの誤配送を招くため，MT は移動前にこれらの binding を無効化しなければならない．



(a) MT1 から CT8 へ BU 送信.　(b) CT8 から MT1 への通信.

図 9 On-Chip Micro Mobility による通信（route optimization 有り）．図 5(d) では，MT1 は CT8 から route optimization されていないパケットを受信したため，図 9(a) では，CT8 に対しても BU を送信している．以降，CT8 は MT1 の CoA を知っているので，CT から MT への通信は Home Core を経由する必要はない．

MT の移動は以下の手順で行われる．

(1) HC に対して BU を送信する（3.3 節と同じ）．
(2) BUL を確認し，BU を送信した CT を調べる．
(3) BU を送信済みの CT に対して，HoA = CoA となる BU を送信する．こうすることで CT は MT 宛てのパケットをすべて HoA（つまり HC）に送るようになる．
(4) MT が移動を開始する．移動が完了したら HC に対して BA を送信する．

Home Core における binding の管理

route optimization を行わないとき（3.3 節）と同じである．

Corresponding Task による binding の管理

HC 同様，CT においても BU を受信したら HoA と CoA の対応付けを BC に保存する．

Mobile Task から Corresponding Task への通信

route optimization を行わないとき（3.3 節）と同じである．

Corresponding Task から Mobile Task への通信（Route Optimization 後）

ここでは，MT の移動後に CT に対しても BU が送信されているので CT は MT の現在の CoA を知っている．このとき，CT から MT への通信は以下の手順で行われる．

(1) CT において，MT の HoA 宛てにパケットを生成する．
(2) CT において，BC をひいて，HoA から対応する CoA を検索する．
(3) ここでは CT の BC に MT の binding が登録されている場合を考えているので，HoA に対応する CoA が見付かる．この場合，CT は HoA 宛てのパケットの宛先を MT の現在の CoA に書き換え，HoA ではなく直接 CoA 宛てに送信する．

(4)　パケットが MT の CoA に届く．

このように CT の BC に宛先 MT の binding が登録されていれば，HC を経由する必要はなく，CT と MT が直接通信できる（**図 9(b)**）．

このような通信を「route optimization された通信」と呼ぶ．route optimization によって，無駄な非最短経路を減らし，また，HC の負荷を抑えることができる．

4. On-Chip Network Mobility

本章では On-Chip Network Mobility を提案する．これは On-Chip Micro Mobility を複数コア単位の移動に拡張したものである．

実際，メニーコアにおいて，近接コア同士の通信のほうが遠く離れたコア同士の通信より圧倒的に多いと考えられる．このような場合，タスク単体を移動させるよりは，近傍コアで動作する複数のタスクをまとめて移動させたほうが効率が良い．On-Chip Network Mobility はこのような用途に適している．

まず，4.1 節で On-Chip Network Mobility が対象とする階層構造を持ったオンチップネットワークトポロジについて説明する．4.2 節では，3 章で述べた On-Chip Micro Mobility と Network Mobility との違いをまとめる．

4.1 階層構造を持つトポロジ

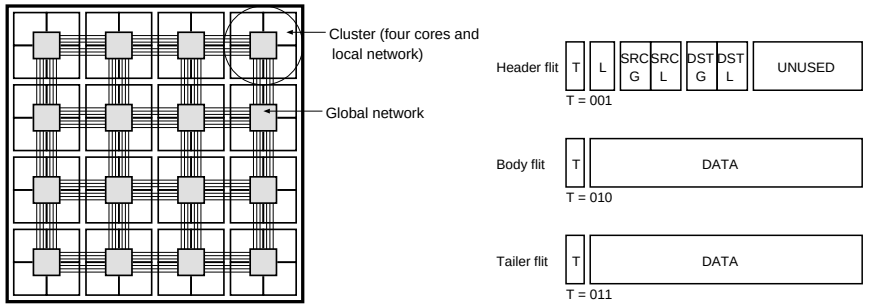
NoC 向けに階層構造を持つ様々なネットワークトポロジが提案されている[9),10)]．階層構造を持つトポロジの非常にシンプルな例を **図 10** に示す．この例では，クラスタ内のコア同士はクロスバで密に結合されているため，隣接コア間の通信を効率良く処理できる．このように階層構造を持ったトポロジは，「遠く離れたコア同士の通信より隣接コア同士の通信のほうが多い」というトラフィックパターンの局所性を活かすことができる．

次に **図 11** を用いて，階層構造を持った NoC におけるタスクの移動について考える．前述したとおり，近接間通信が多い場合，タスク単体を移動させるよりは，近傍コアで動作する複数のタスクをまとめて移動させたほうが効率が良い．この例では，クラスタ内の 4 個のコアで動作しているタスクが，まとめて他のクラスタに移動している．次節では，このような移動に対処にするための On-Chip Network Mobility プロトコルを説明する．

4.2 On-Chip Network Mobility プロトコル

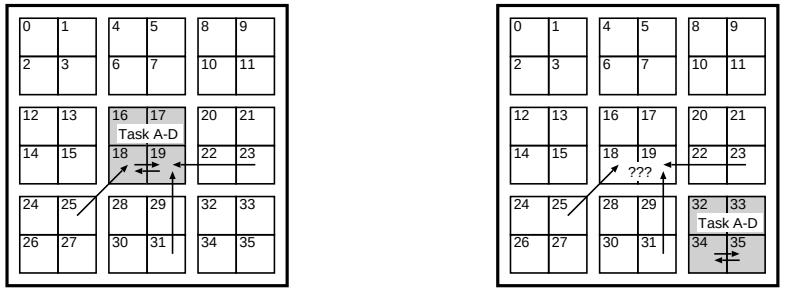
On-Chip Micro Mobility プロトコルと On-Chip Network Mobility プロトコルの主な違いはパケットのフォーマットである．binding の通知（BU および BA），binding の管理機構（BC および BUL），route optimization などについては On-Chip Micro Mobility プロトコルと同じである．

階層構造を持つ NoC では，**図 10(b)** に示すように，パケットヘッダの SRC と DST フィールドは，Global（G）フィールドと Local（L）フィールドに分かれている．On-Chip Network Mobility ではクラスタ単位で移動するため，アドレスの L フィールドは移動の影響を受けない．移動によって変化するのは G フィールドのみである．



(a) ネットワーク構造.　(b) パケットフォーマット.

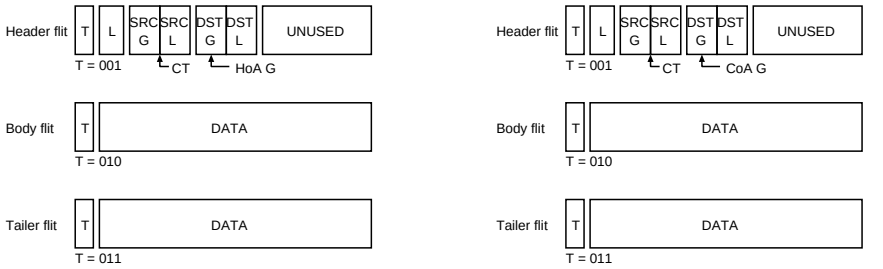
図 10 階層構造を持つ NoC の例．4 個のコアが 1 つのスイッチで密に結合され，1 つクラスタを形成している．クラスタ内の結合をローカルネットワークと呼ぶ．この例ではチップ内に 16 個のクラスタが存在し，クラスタ同士は 2 次元メッシュ状にグローバルなネットワークで結合されている．パケットのアドレスフィールド（SRC と DST）は，Global（G）フィールドと Local（L）フィールドに分かれている．



(a) Task A，B，C，D の移動前.　(b) Task A，B，C，D の移動後.

図 11 クラスタ内の複数タスクの移動に伴う問題．この例ではコア 16-19 の上で Task A-D が動作しており，コア 23，25，31 上のタスクと通信をしている．その後，Task A-D はコア 32-35 へ移動する．Task A-D の移動後，それまで Task A-D と通信していたコア 23，25，31 上のタスクたちは通信を継続できなくなる．

図 12 に On-Chip Network Mobility において route optimization を行わないときの CT から MT へのパケットを示す．route optimization を行うときは，CT は MT の CoA を知っているため，CT は最初から **図 12(b)** のフォーマットでパケットを送信できる．



(a) パケットフォーマット（CT から HC）.　(b) パケットフォーマット（HC から MT）.

図 12 On-Chip Network Mobility において，route optimization しないときの CT から MT へのパケットフォーマット．パケットは HC で中継される．CT から HC までの転送では MT の HoA の上位フィールドが DST G であり，HC から MT までの転送では MT の CoA の上位フィールドが DST G となる．

5. On-Chip Micro Mobility の設計

本章では，On-Chip Micro Mobility プロトコルの NoC のハードウェアへの実装方法について検討する．

ここでは On-Chip Micro Mobility の機能をネットワークインタフェイス（NI）に実装し，ルータ本体には手を加えないようにする．**図 5.2** に On-Chip Micro Mobility をサポートした NI のアーキテクチャを示す．NI に必要な変更点は 1) BC の実装，2) BUL の実装，3) Mobile Task Proxy 機構である．以降の節ではそれぞれについて説明する．

5.1 Binding Cache の実装

BC は CT および HC 側の機能である．NI でパケットを生成する度に，パケットの DST（HoA）をキーに BC をひく．もし，対応する CoA が BC にあれば DST を CoA に書き換えてパケットを送信する．無ければ HoA 宛てのままパケットを送信する．BC の構造は図 8(a) に示したとおりである．

NI に BC を実装しないと，外部コアを訪問中の MT 宛てのパケットは，すべて HC を経由することになり，スループット性能や通信遅延に影響が生じてしまう．

5.2 Binding Update List の実装

BUL は MT 側の機能である．現在の CoA と HoA の binding を持っている CT にマークを付ける．つまり，BU を送信する度に BUL を更新する．BUL の構造を 図 8(b) に示す．

タスクが別のコアに移動する直前に，BUL にリストされている CT に対し，binding を無効化するための BU（CoA = HoA）を送信する．こうすることで移動直後の誤配送を防ぐ．

5.3 Home Core による Mobile Task Proxy 機構

MT Proxy 機構は HC 側の機能である．NI でパケットを受信する度に DST を調べる．その DST が自分が管理している MT の HoA であれば，DST を MT の現在の CoA に書き換えたうえで MT に向けて転送する．

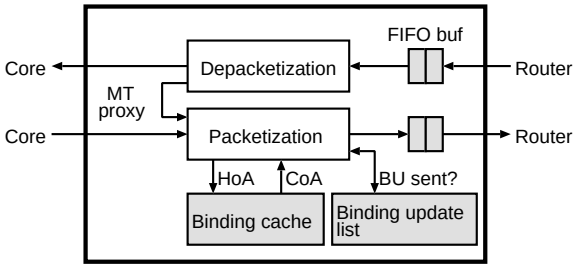


図 13 On-Chip Micro Mobility をサポートしたネットワークインタフェイス．CT 側の機能として，パケットを生成する際には BC をひき，宛先の HoA から CoA を調べる．MT 側の機能として，CT に対して BU を送信する度に BUL を更新する．さらに HC 側の機能として MT Proxy を行う．MT Proxy によって，この HoA を持つ MT 宛てに来たパケットを現在位置（CoA）へと転送する．

HC によるパケットの転送時には，デッドロックを防ぐために，これまでと異なる仮想チャネルを使用しなければならない．通常，NoC 内の通信には次元順ルーティングやターンモデルなどのデッドロックフリー・ルーティング[2] が用いられる．route optimization されていないパケット転送の場合，1) CT から一度 HC に到達し，2) HC から MT の CoA に転送される．このような転送では，1) と 2) の間のターンにおいて循環依存，つまり，デッドロックが生じる可能性がある．このようなデッドロックを防ぐためにはいくつか方法があるが，ここでは仮想チャネルを切替えることで 1) と 2) の間の循環依存を断ち切ることにする．

6. 予備評価

本章では On-Chip Micro Mobility プロトコルの予備評価として，route optimization の効果を示し，そのための追加ハードウェアコストを見積もる．

6.1 Route Optimization の効果

ここでは，route optimization するときとしないときの平均ホップ数とスループットの期待値を見積もる．

与えられたトポロジにおいて最短ルーティングを用いた際の平均ホップ数を H_{min} とする．route optimization する場合，MT-MT 間は常に最短経路で通信できるため，平均ホップ数は H_{min} のままである．一方，route optimization しない場合，MT-MT 間の通信は常に HC を経由することになる．このときの MT-MT 間のホップ数は HC の位置に依存するが，HC の位置が一様に分散していると想定するとホップ数は $2H_{min}$ となる．つまり，route optimization しないと平均ホップ数が倍になり，通信遅延や消費エネルギーが大幅に増加してしまう．

ネットワークの理想的なスループットは次式で計算できる[4]．

$$\Theta_{ideal} \leq \frac{bC}{NH} \quad (1)$$

ただし，b をチャネルの帯域，C をネットワークの総チャネル数，N をノード数，H をホップ数とする．route optimization しない場合の平均ホップ数（H）は通常の倍となるため，期待されるスループットは半分に落ち込んでしまう．

6.2 追加のハードウェアコスト

5 章で述べたとおり，On-Chip Micro Mobility プロトコルは NI に実装されるため，BC および BUL が追加ハードウェアとして必要となる．route optimization を行わないなら BC および BUL は不要ではあるが，前節で示したように route optimization を用いないと平均ホップ数およびスループットが大幅に悪化してしまう．

ここでは route optimization のためのハードウェアオーバヘッドとして BC と BUL のメモリ量を計算する．これらのメモリ量をオンチップルータのバッファ量と比較することで，追加コストが NoC 全体に与える影響を見積もる．

ルータ 1 個当たりのバッファ量，BC および BUL のビット数は以下の式で計算できる．

$$BUF_{bit} = pv \times wd \quad (2)$$

$$BC_{bit} = (\log_2 N + 1)N \quad (3)$$

$$BUL_{bit} = N \quad (4)$$

ただし，p をルータのポート数，n を仮想チャネル数，w をフリット幅，d をバッファの深さ（フリット数），N をルータの個数とする．ここでは 2 次元トーラストポロジにおける一般的な仮想チャネルルータを想定し，$p = 5$，$v = 2$，$w = 64$，$d = 4$ とした[10]．

$N = 16$ のとき，BUF_{bit} の 2,560 bit に対し，BC と BUL のビット数は 96 bit となり，オーバヘッドは高々 3.8% である．ところが，これが $N = 64$ で 20.0%，$N = 256$ で 100.0% と膨れ上がる．つまり，256 コアのように非常に多いコア数では，どこかのコアで動作しているタスクの現在位置をすべて把握しようとすると BC が巨大化してしまう．この問題について我々は以下の a) と b) の 2 つの解決方法を検討している．

a) On-Chip Network Mobility を用い，4 タスク程度のクラスタ単位で移動を行う．$N = 256$ の場合，管理しなければいけないクラスタの数は 64 個に減るので，ハードウェアオーバヘッドは 20.0% 程度に抑えられる．

b) そもそも単一タスクが一度に通信する相手は非常に少ない．また，通信相手が BC に登録されていないと，HC 経由の通信になりスループットが落ちるが，通信自体は可能である．そこで，BC のエントリ数を 4〜8 個程度に抑えてしまう．この場合，HoA をキーに BC から CoA を 1 サイクルで得るのは難しいが，パケット生成には通常複数サイクルかかるので，影響を隠蔽できると考えられる．

7. まとめと今後の課題

本論文ではあるコアで動作していタスクを別のコアへ移動させることを可能とする On-Chip Micro Mobility プロトコルを提案した．これにより，Mobile Task と呼ばれるタスクがどのコア上で動作していたとしても，Mobile Task ごとに固有の識別子でアクセスできる．つまり，この識別子宛てにパケットを送信すれば，Mobile Task がどのコア上で動作していたとしても必ずこのタスクにパケットが届く．さらに，このアイディアを拡張して，タスク単体を移動させるのではなく近傍コアで動作する複数のタスクをまとめて移動させた場合にも移動透過性、および通信可能性を保証する On-Chip Network Mobility プロトコルを示した．

現状では，本プロトコルを設計して予備評価を示しただけである．今後の課題としては，

- On-Chip Network Mobility の機能を NI 回路にハードウェアとして実装し，面積および消費エネルギーを見積もる．
- On-Chip Network Mobility プロトコルを GEMS などのフルシステム・シミュレータ上に実装し，プロセッサ間でのタスクの移動を含めた評価を行う．

参 考 文 献

1) Vangal, S.R., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., Finan, D., Singh, A., Jacob, T., Jain, S., Erraguntla, V., Roberts, C., Hoskote, Y., Borkar, N. and Borkar, S.: An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS, *IEEE Journal of Solid-State Circuits*, Vol.43, No.1, pp.29–41 (2008).
2) Dally, W.J. and Towles, B.: Route Packets, Not Wires: On-Chip Interconnection Networks, *Proceedings of the Design Automation Conference (DAC'01)*, pp.684–689 (2001).
3) Benini, L. and Micheli, G.D.: *Networks on Chips: Technology And Tools*, Morgan Kaufmann (2006).
4) Dally, W.J. and Towles, B.: *Principles and Practices of Interconnection Networks*, Morgan Kaufmann (2004).
5) Murali, S., Theocharides, T., Vijaykrishnan, N., Irwin, M.J., Benini, L. and Micheli, G.D.: Analysis of Error Recovery Schemes for Networks on Chips, *IEEE Design and Test of Computers*, Vol.22, No.5, pp.434–442 (2005).
6) Perkins, C.E. et al.: *IP Mobility Support for IPv4* (2002). RFC 3344.
7) Arkko, J., Devarapalli, V. and Dupont, F.: *Mobility Support in IPv6* (2004). RFC 3775.
8) Devarapalli, V., Wakikawa, R., Petrescu, A. and Thubert, P.: *Network Mobility (NEMO) Basic Support Protocol* (2005). RFC 3963.
9) Balfour, J.D. and Dally, W.J.: Design Tradeoffs for Tiled CMP On-Chip Networks, *Proceedings of the International Conference on Supercomputing (ICS'06)*, pp.187–198 (2006).
10) Matsutani, H., Koibuchi, M., Hsu, D.F. and Amano, H.: Fat H-Tree: A Cost-Efficient Tree-Based On-Chip Network, *IEEE Transactions on Parallel and Distributed Systems* (2009). (to appear).