Vol.2014-ARC-213 No.16
Vol.2014-HPC-147 No.16
2014/12/9

FOLCS: FPGAを用いた軽量サイクル・アキュレートNoCシミュレータ

成子　貴洋[1,a)]　平木　敬[1,b)]

概要：マルチコア・メニーコアへの潮流は，コンピュータアーキテクチャのシミュレーションを時間的により困難なものとしている．シミュレーションを実行するホストのコアの速度向上が緩やかであることと，シミュレータの並列化の難しさが原因である．FPGA によるシミュレーションの実行・アクセラレーションは，シミュレーション時間を短縮する有力な手段である．FPGA を用いることの制約は，使用可能なリソースに限りがある点である．本稿は，チップ上のインターコネクションネットワーク (Network on Chip, NoC) に焦点を絞り，FPGA 上で動作する軽量な NoC シミュレータ FOLCS(Flit-Oriented Lightweight Cycle-accurate network Simulator) を提案する．FOLCS は，既存手法よりも少ない FPGA リソースでサイクルアキュレートな NoC シミュレーションを実現する．

1. はじめに

コンピュータアーキテクチャの研究において，シミュレータはアイデアを評価するための重要な手段である．特に，実装が容易なソフトウェアシミュレータが広く用いられている．マルチコアへの舵を取る以前は，評価対象の規模と並行して，シミュレーションを実行するホストのシングルスレッド性能が向上していた．評価対象とホストの足並みが揃っていたために，現実的な時間でソフトウェアによるシミュレーションが可能であった．しかし，近年のマルチコア・メニーコアへの潮流は，評価対象の複雑化とホストのシングルスレッド性能の停滞を招き，ソフトウェアによるシミュレーションを時間的に困難なものとしている．シミュレータの並列化 [4] はこの問題に対する解法の一つであるが，細粒度の同期が必要となることから，並列化の効果は限定的である．

このような背景から，アーキテクチャシミュレーションへの FPGA の活用が研究されている．FPGA によるシミュレーションは抽象化の度合いに基づき，2 つのタイプに分類することができる [9]．評価対象と RTL での互換性を有するタイプと，抽象化されたタイプである．前者の例として，Intel®による FPGA で動作可能な Nehalem プロセッサが挙げられる [7]．FPGA 基板上で 4 コア Nehalem プロセッサが 520kHz で動作するため，高速な RTL 検証や，テープアウト前のソフトウェア開発に用いることができる．短所は，デザインが 1 つの FPGA に収まらないため特殊な基板を必要とする点，RTL 互換のため設計前のパラメータ探索には用いることができない点である．抽象化されたタイプの例としては，HAsim[6] や RAMP Gold[8] が挙げられる．このタイプのシミュレータでは，Functional Model と Timing Model が分離され，評価対象のアーキテクチャは機能部の集合として抽象化される．抽象化により，1) 使用 FPGA リソースの削減，2) 仕様変更の簡略化，3) 実行時パラメタによるデザイン探索の高速化，4)RTL コードが存在しない開発前の評価に用いることができる，といった利点が得られる．抽象化に加え，HAsim[6] や RAMP Gold[8] は時分割多重を採用している．機能部の実体を 1 つだけ用意し，その内部状態を時分割で切り替えることで，仮想的に複数個の機能部のシミュレーションを行うのである．この工夫により，必要な FPGA リソースを削減することが可能である．

本稿では，プロセッサに含まれる機能部のうち，インターコネクションネットワーク部 (Network on Chip, NoC) に焦点を絞る．FPGA 上に直接ルータやリンクを構成する手法は，FPGA で動作する最も単純な NoC シミュレータである．この直接的な方法の欠点は，多くの FPGA リソースを消費する点である．例えば，Xilinx Virtex 2 Pro v20 上に構成できるルータ数は 6 つに留まる [5]．DART[10] は NoC を単純なモデルで表現することで，シミュレーション精度の低下と引き換えに，使用 FPGA リソース数の削減を実現している．Wolkotte ら [11] は時分割多重を用い，

[1] 東京大学
Hongo, Bunkyo-ku, Tokyo 7-3-1, Japan
[a)] cinccinaru@is.s.u-tokyo.ac.jp
[b)] hiraki@is.s.u-tokyo.ac.jp

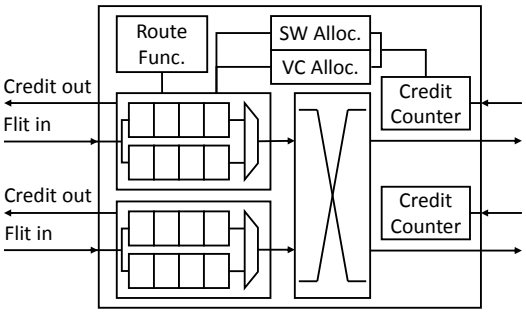

図 1　想定する NoC のルータブロック図.

サイクルアキュレートを保証しつつ，使用 FPGA リソース数の削減を図った．各々が独立して動作するコアと異なり，NoC は隣接するルータ間に相互作用が存在するため，時分割多重を行うには工夫を要する [6]．Wolkotte らは，NoC の状態が収束するまで反復してルータの状態計算を行う [12] ことで，ルータ間の相互作用に対処している．この方法の欠点は，NoC の状態を退避復元するために，広い Block RAM 帯域が必要となる点である．仮想チャネルアロケータを省略し，かつ簡略化したスイッチアロケータを用いる場合でも，2112bit の Block RAM 帯域が必要である [11].

本稿は，FOLCS(Flit-Oriented Lightweight Cycle-accurate network Simulator) を提案する．FOLCS は NoC を，ルータの状態遷移でなく，フリットの状態遷移として扱うことで，サイクルアキュレートを保ちつつ，状態の退避復元に要する帯域を削減する．

2. NoC

FOLCS について述べる前に，想定する NoC の構成要素と動作について簡単に解説する．本稿では，仮想チャネルとワームホールルーティングを採用した 5 段パイプラインの NoC[3] を想定する．図 1 はルータのブロック図である．仮想チャネルとスイッチのアロケータは，ラウンドロビン方式を用いたものを想定する．フロー制御はクレジットにより行う．パケットはフリットと呼ばれる単位に分割された上でネットワーク上を移動する．各ルータは以下の 5 つの処理を順にフリットに施す．

BW　フリットのキューへの挿入と，出力ポートの計算
VA　次ルータにおける仮想チャネルの割り当て
SA　クロスバースイッチの割り当て
ST　クロスバースイッチの通過
LT　ルータ間リンクの通過

3. FOLCS

3.1 概要

FOLCS は NoC のシミュレーションをルータ単位でなく，フリット単位で行う．FOLCS のブロック図を図 2 に示す．灰色の長方形は FPGA の Block RAM に割り当てる

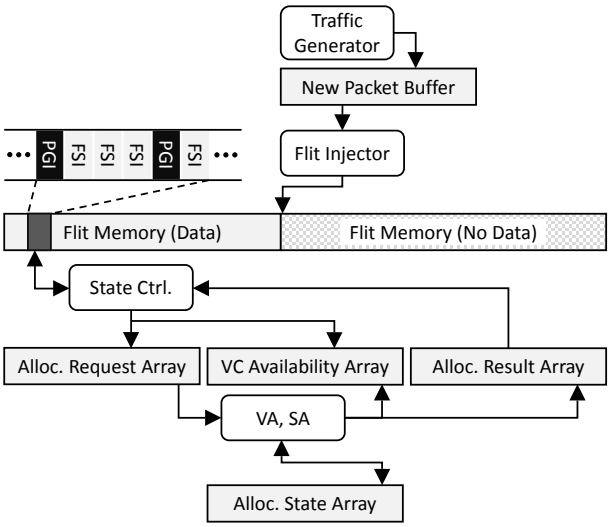

図 2　FOLCS のブロック図.

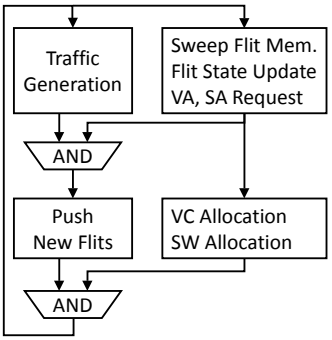

図 3　FOLCS の動作フロー.

配列構造を表す．各配列のサイズと役割を表 1 にまとめる．角丸の長方形は配列内データの更新・追加を行う機構である．Traffic Generator は特定のトラフィックパターンに基づいたパケットを生成し，New Packet Buffer に追加する．Flit Injector は生成したパケットを Flit Memory に挿入する．State Ctrl. は Flit Memory 内に格納されたフリットの状態と Alloc. Result Array 内の割り当て結果から，新たな状態を計算し Flit Memory に上書きする．同時に，必要であれば，Alloc. Request Array や VC Availability Array の更新を行う．VA,SA は，Alloc. Request Array 内の割り当て要求，VC Availability Array 内の空き仮想チャネル情報，Alloc. State Array 内のアロケータの状態に基づき，仮想チャネルとクロスバースイッチの割り当てを行う．割り当て結果は Alloc. Result Array に，アロケータの新しい状態は Alloc. State Array に，それぞれ書き込まれる．新たに仮想チャネルを割り当てた場合は，VC Availability Array 内の対応するビットをクリアする．

FOLCS の動作フローを図 3 に示す．評価対象上の 1 サイクル (以下シミュレーションサイクルと記す) のシミュレーションは，4 つのブロックから成る．始めに行うのは，新しいパケットの生成と Flit Memory の更新であり，Traffic Generator と State Ctrl. がそれぞれを担う．新しいパケッ

表 1　FOLCS が有する各配列のサイズと役割．N, R, P, V, L はそれぞれ，ノード数，ルータ数，ルータのポート数，ポートあたりの仮想チャネル数，最大パケット長，を表す．

名称	幅	深さ	役割
New Packet Buffer	35bit	N	ネットワークにインジェクションするパケットを保持
Flit Memory	36bit	$R \times P \times V \times (L+1)$	ネットワーク内のフリットの状態を保持
Alloc. Request Array	100bit	R	仮想チャネルやクロスバースイッチの割り当て要求を管理
VC Availability Array	20bit	R	仮想チャネルの使用・未使用を管理
Alloc. Result Array	80bit	R	仮想チャネルやクロスバースイッチの割り当て結果を管理
Alloc. State Array	165bit	R	アロケータの状態を保存

トの生成にはノード数に等しい FPGA 上のサイクル (以下 FPGA サイクルと記す) を要する．Flit Memory には有効なデータが前から詰めて保存されるため，Flit Memory の更新処理に要する FPGA サイクルは Flit Memory 内の有効なデータ数に等しい．Flit Memory の更新が完了し次第，VA,SA が起動され，仮想チャネルとクロスバースイッチの割り当てが行われる．現在の実装では，全てのルータに対して割り当て計算を行うため，この処理はルータ数に等しい FPGA サイクルを要する．New Packet Buffer 内に生成されたパケットの Flit Memory への挿入は，パケット生成と Flit Memory の更新の両方が終了した後に行われる．この処理は挿入するデータ数に等しい FPGA サイクルを要する．割り当て計算とパケットの挿入が完了すると，シミュレーションサイクルを進め，上記の処理を繰り返す．

3.2 Flit Memory

Flit Memory に格納されるデータには，PGI(Packet General Information) と FSI(Flit Specific Information) の 2 種類が存在する．1 パケットの情報は，1 つの PGI と，それに続くいくつかの FSI により構成される．PGI は，ソース・デスティネーションノードの番号やパケット生成時のタイムスタンプといった，パケットの静的な情報を持つ．FSI は，シミュレーションサイクルの経過とともに変化する，フリット固有の情報を持つ．具体的には，Head/Tail フリットか否か，現在のルータ番号，ポート番号，仮想チャネル番号，ステート，といった情報が含まれる．PGI の後にパケット長に等しい数の FSI が続き，i 番目の FSI はパケットの i 番目のフリットに対応する．

フリットが目的のノードまで到達した場合は，Flit Memory から当該 FSI を削除する必要がある．要素の削除は，書き込みアドレスのオフセットをインクリメントし，後続の書き込みデータを不要になった要素に上書きすることで実現する．有効なデータが Flit Memory の先頭方向に詰められた状態が保たれるため，Flit Memory の走査は先頭から有効なデータが存在する範囲のみを行えばよい．

最大 256 ノード 5 ポート 4 仮想チャネルをサポートする現在の実装では，PGI のデータ幅は 33bit，FSI のデータ幅は 20bit である．しかし，Virtex-6 FPGA の Block RAM が 36bit 幅であることと，将来の拡張性を考え，Flit Buffer の幅は 36bit を確保している．深さは最悪の場合を考慮し，[ルータ数]×[ポート数]×[仮想チャネル数]×[最大パケット長 +1] を用意している．

3.3 アロケータ関連配列

アロケータ関連の配列 (Alloc. Request Array, VC Availability Array, Alloc. Result Array) の各要素は，1 つのルータに対応する．各要素はルータ内の各入力ポートに対応する領域に分割でき，その領域はさらに，各仮想チャネルに対応する領域に分割できる．つまり，これらの配列は [ルータ数]×[ポート数]×[仮想チャネル数] の 3 次元配列と見なせ，以下では 3 次元配列として見た時の要素を小要素と呼ぶ．

Alloc. Request Array の小要素には，仮想チャネル割り当て要求ビット，クロスバースイッチ割り当て要求ビット，出力ポート番号が含まれる．VC Availability Array の小要素は，当該仮想チャネルの利用可否を表すビットを持つ．Alloc. Result Array の小要素には，仮想チャネル割り当て結果ビット，割り当てられた仮想チャネル番号，クロスバースイッチ割り当て結果ビットが含まれる．割り当てられた仮想チャネル番号は，対応する仮想チャネル内のパケットがルータを去るまで保持する必要があるため，仮想チャネルの割り当てが行われた時のみ更新される．

3.4 State Controller

State Ctrl. は Flit Memory を先頭から走査しながら，フリットの状態を更新する．Flit Memory から読み込んだデータが PGI である場合は，PGI の情報をレジスタに記録し，次の PGI を読み込むまで保持する．

フリットのステート更新と割り当て要求が発生する条件を図 4 に示す．丸囲いはフリットのステートを，四角囲いは Alloc. Request Array や VC Availability Array の更新を，それぞれ表す．

新たなステートが BW である場合には，フリットの位置に関する情報を更新する．新しいルータ番号とポート番号は，ルーティング関数により計算される．ルーティング関数は，現在のルータ番号とソース・デスティネーションノード番号を引数に取り，出力ポート番号，次のルータ番

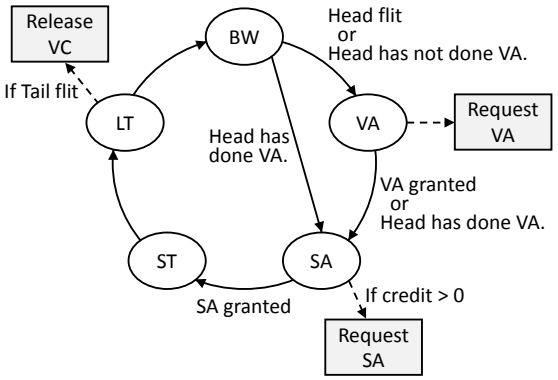

図 4　ステート更新と割り当て要求の条件.

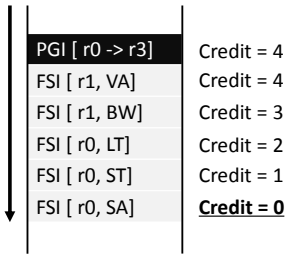

図 5　Flit Memory の走査過程でクレジットの計算が行われる例.

号，入力ポート番号を返す関数である．ルーティング関数を適切に設定することで，任意のトポロジやルーティング規則を実現できる．ルーティング関数はハードウェアで実装されるため，FPGA のプログラムファイル生成後の関数変更は不可能である．新しい仮想チャネル番号は，Alloc. Result Array に保存されている番号を用いる．

古いステートが LT であり，かつ次ルータ番号の計算結果がネットワーク外ノードである場合は，FSI の削除を行う．上述の通り，FSI の削除は書きこみアドレスのオフセットをインクリメントすることにより実現される．フリットが末尾フリットである場合は，オフセットに 2 を足して，FSI と PGI を同時に削除する．さらに，PGI がトップモジュールに送られ，シミュレーションの統計情報が更新される．

仮想チャネルの割り当ては，先頭フリットのみが行う処理である．先頭フリットが VA ステートである場合には，Alloc. Request Array に仮想チャネルの割り当て要求と出力ポート番号を記録する．先頭フリットが仮想チャネルの割り当てを完了済みのルータでは，後続のフリットは VA ステージを飛ばして SA ステージに進むことができる．仮想チャネルの割り当てが済んでいるか否かは，Flit Memory を走査する過程で判定できる．走査の過程で先頭フリットの現在位置 r_0(ルータ番号) とステート s_0 をレジスタに記憶させる．現在位置が r_i の後続フリットに関して，$r_i \neq r_0$ であるか，s_0 が SA より先のステートである場合には，仮想チャネルの割り当ては完了済みであると判断できる．

フリットが SA ステートである場合には，Alloc. Request Array にクロスバースイッチ割り当て要求を記録する．パケットが入力バッファのサイズよりも長い場合には，フロー制御を行う必要がある．FSI が先頭フリットから順に並んでいるため，クレジットは Flit Memory を走査する過程で計算できる．例を図 5 に示す．クレジットの計算結果が 0 の場合には，フリットのステートが SA であっても，クロスバースイッチの割り当て要求を見送る．

末尾フリットが LT ステートである場合には，VC Availability Array の対応するビットをセットし，仮想チャネルを解放する．対応する小要素のインデックスは，一つ前のルータ番号と出力ポートであるため，ルーティング関数の逆関数を計算する必要がある．ルーティング関数と同様に，逆関数はハードウェアにより実装される．

3.5 Traffic Generator

ネットワークの評価に用いるトラフィックの生成は，Traffic Generator が行う．現在の実装では，ベルヌーイ過程に基づく一様乱数トラフィックパターンのみをサポートする．VHDL コードを数行書き換えることで，他のトラフィックパターンにも対応可能である．インジェクションレートは，シミュレーション開始時にホスト PC から RS232C 経由で指定する．ベルヌーイ過程に用いる擬似乱数は，16bit の LFSR により生成する．デスティネーションノードの決定に用いる擬似乱数は，32bit の LFSR により生成する．

Flit Injector は，New Packet Buffer 内のパケット情報を PGI と FSI の列に展開し，Flit Memory にプッシュする．FSI のルータ・ポート番号領域には，パケットを生成したノードが繋がっているルータの番号 r とポート番号 p を記録する．仮想チャネルは，ルータ r の p 番ポートから空いている仮想チャネルを選択して割り当てる．外部ノードに繋がったポートにおける仮想チャネルの使用・未使用は，VC Availability Array と同様の配列で管理する．

レイテンシの測定においては，キューイング遅延を考慮することが重要である [3]．しかし，インジェクション待機中のパケットを保持するために，無限長のキューを用意することは非現実的である．生成済みのパケットをキューに保持する代わりに，ノード毎に時刻を管理する手法 [3] を用いる．パケットのインジェクションを行わない場合や，インジェクションが成功した場合には，ノードの時刻をインクリメントする．パケットのインジェクションが失敗した場合には，ノードの時刻を進めずに，次のシミュレーションサイクルでインジェクションの再試行を行う．ノードの時刻は，幅が 16bit で深さがノード数の配列により管理される．

3.6 統計情報

現実装で集める統計情報は，イジェクトされたパケット数と，それらのレイテンシの合計の 2 種類である．いずれの値も，32bit のレジスタで管理される．これらの情報は，

表 2　XC6VLX240T における 256 ノード FOLCS の使用リソース数.

	Used	Available	Utilization
Slices	1774	37680	4%
Registers	2341	301440	1%
6-input LUT	5318	150720	3%
36Kb Block RAM	45	416	11%

シミュレーション終了後に RS232C 経由でホスト PC に送られる．ホスト PC は，レイテンシの合計をパケット数で割り，平均レイテンシを計算する．

4. 性能評価

4.1 実装コスト

256 ノード，5 ポート，4 仮想チャネルのシミュレーションが可能な FOLCS を，Xilinx Virtex-6 FPGA ML605 評価キット [13] に実装し，使用 FPGA リソース数の評価を行った．トポロジは 16 × 16 の 2 次元メッシュを想定する．ML605 評価キットは，37,680 個のスライスと 416 個の 36Kbit Block RAM を持つ XC6VLX240T-1FFG1156 FPGA を搭載する．デザインの論理合成と配置配線には，Xilinx ISE 14.4 を用いた．

配置配線後の，スライスと Block RAM の消費量を表 2 に示す．スライスよりも Block RAM を多く必要とするデザインであることが分かる．使用 Block RAM 45 個のうち，30 個は Flit Memory が占める．Flit Memory は最悪の場合を考慮して確保しているが，実用上は半分程度の深さでも十分であると考えられる．例えば，パケット長 5 フリット，入力ポートバッファの深さ 3 フリット，インジェクションレート 0.5 packet/node/cycle で 5000 サイクルのシミュレーションを行った場合，Flit Memory の使用量は 30K ワード中，最大でも 10547 ワードであった．インジェクションレート 0.5 packet/node/cycle は飽和スループット 0.05 packet/node/cycle よりも充分大きな値である．実アプリケーションによるトラフィックの場合には，Flie Memory の使用量はさらに少ないと予想される．Block RAM に確保する Flit Memory の領域を 10K ワード程度に削減し，残りの領域を外部メモリに確保する仕組みを導入すれば，FOLCS の実装をより軽量にできると考えられる．この点の改良は今後の課題である．

4.2 妥当性検証

FOLCS の妥当性を検証するために，ソフトウェア NoC シミュレータである GARNET[1] と結果の比較を行う．シミュレーション実行環境を表 3 にまとめる．評価に用いるネットワーク構成を表 4 に示す．GARNET は，フル・システム・シミュレータ gem5[2] に組み込まれており，ネットワークのシミュレーションは特別なキャッシュ・コヒーレンス・プロトコルを用いて行われる．GARNET では 65 ノード以上のシミュレーションを実行できないため，妥当性検証は 64 ノードで行う．パケット生成からインジェクションまでのレイテンシを揃えるため，以下で示す GARNET のレイテンシは，GARNET のデフォルト実装によるレイテンシから 4 を減算した値である．

表 3　シミュレーション実行環境.

FOLCS	
動作周波数	66MHz
RS232C ボーレート	115200 bps
GARNET	
CPU	Intel® Core™ i7-4770
Memory	8GB PC3-12800 × 4
OS	Ubuntu 12.04

表 4　GARNET との比較で用いるネットワーク構成.

ノード数	64
ポート数	5
仮想チャネル数	4
入力ポートバッファの深さ	3
トポロジ	8×8 メッシュ
パケット長	5 フリット
シミュレーション時間	5000 サイクル

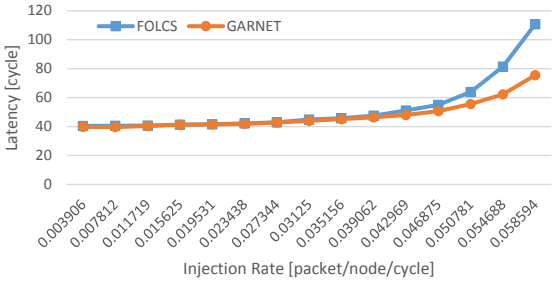

図 6　8×8 メッシュネットワークに 5 フリットパケットをインジェクションした際の，負荷-レイテンシグラフ.

FOLCS と GARNET によるレイテンシの評価結果を図 6 に示す．レイテンシが 0.039062 packet/node/cycle までの領域では，FOLCS の結果は GARNET の結果の 3%以下の差異に収まっている．それ以上の領域では，インジェクションレートの増加と共に結果に乖離が生じている．アロケータや乱数生成における両シミュレータ間の違いが原因であると予想している．本稿執筆時点で原因を究明中である．

4.3 シミュレーション実行時間

FOLCS と GARNET のシミュレーション実行時間の比較を行う．シミュレーション実行環境とネットワーク構成は，妥当性検証と同様に，表 3 と表 4 の通りである．

図 7 に，各インジェクションレートにおける FOLCS と GARNET のシミュレーション実行時間を示す．GARNET と比べ，FOLCS を用いることで 18 倍から 129 倍の高速

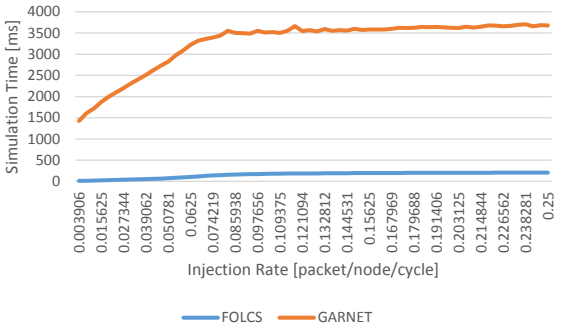

図 7 8×8 メッシュネットワークに 5 フリットパケットをインジェクションした際の，負荷-実行時間グラフ．

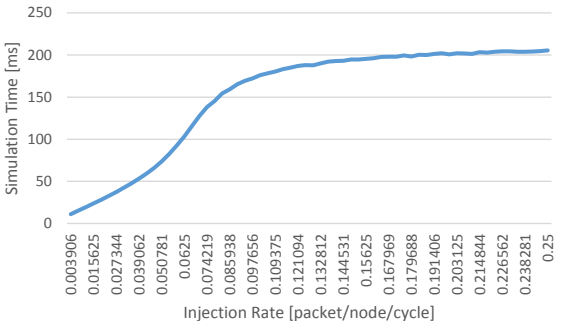

図 8 FOLCS の負荷-実行時間グラフ．

化が得られている．高速化は低インジェクションレートの領域で大きく得られ，インジェクションレートの増加とともに高速化は減少する．FOLCS のみの実行時間の結果を図 8 に示す．インジェクションレートの増加とともに，シミュレーション実行時間は単調増加している．トラフィックの増加に伴い，Flit Memory 内の走査対象データ数が増加するためである．0.191406 packet/node/cycle 以上のインジェクションレートでは，実行時間がほぼ収束している．この領域ではネットワークが飽和しており，Flit Memory 内のデータ数は頭打ちとなるからである．

5. まとめと今後の課題

本稿では, FPGA 上で動作する NoC シミュレータ FOLCS を提案した．FOLCS は，サイクルアキュレートなシミュレーションを軽量な FPGA リソースで実現することに重点を置く．最大 256 ノードのシミュレーションが可能な FOLCS を XC6VLX240T-1FFG1156 FPGA に実装する場合，スライス使用率は 4%，Block RAM 使用率は 11%である．ソフトウェア NoC シミュレータ GARNET[1] と FOLCS の出力結果の比較から，低インジェクションレートでは両シミュレータの結果が 3%以下の差異に収まることが確かめられた．FOLCS を用いることで，GARNET と比べ，18 倍から 129 倍のシミュレーション高速化が達成された．

低インジェクションレートでは FOLCS と GARNET の出力結果の一致が確認できたものの，飽和スループットに近い領域では両シミュレータの結果に乖離が生じた．この乖離をもたらす原因については，本稿執筆時点で究明中である．

FOLCS を用いることで，Wolkotte らの手法 [11] や DART[10] と比較して，使用 FPGA リソース数の削減が達成された．しかし，依然 11%の Block RAM を消費している．使用 Block RAM 45 個のうち 30 個は Flit Memory が占めるが，実用上 Flit Memory が上限まで使用されることはないと考えられる．Flit Memory の一部を外部メモリに配置すれば，使用 Block RAM 数の大幅な削減が期待できる．この点の改良は，今後の課題とする．

現在の実装では，仮想チャネルとクロスバースイッチの割り当てを，全てのルータに対して毎シミュレーションサイクル行う．この方法は実装が単純であるという利点を有するが，常にルータ数に等しい FPGA サイクルを要する．割り当て要求を持つルータに対してのみ割り当て計算を行う最適化を導入することで，割り当て計算に要する時間を短縮できる．すでに現在の実装は，割り当て要求の有無を管理する配列を有しているため，このような最適化の導入は可能であると考えている．今後，対応を行う予定である．

参考文献

[1] Agarwal, N., Krishna, T., Peh, L.-S. and Jha, N.: GARNET: A detailed on-chip network model inside a full-system simulator, *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pp. 33–42 (online), DOI: 10.1109/ISPASS.2009.4919636 (2009).

[2] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The Gem5 Simulator, *SIGARCH Comput. Archit. News*, Vol. 39, No. 2, pp. 1–7 (online), DOI: 10.1145/2024716.2024718 (2011).

[3] Dally, W. and Towles, B.: *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2003).

[4] Donald, J. and Martonosi, M.: An Efficient, Practical Parallelization Methodology for Multicore Architecture Simulation, *Computer Architecture Letters*, Vol. 5, No. 2, pp. 14–14 (online), DOI: 10.1109/L-CA.2006.14 (2006).

[5] Genko, N., Atienza, D., De Micheli, G., Mendias, J., Hermida, R. and Catthoor, F.: A complete network-on-chip emulation framework, *Design, Automation and Test in Europe, 2005. Proceedings*, pp. 246–251 Vol. 1 (online), DOI: 10.1109/DATE.2005.5 (2005).

[6] Pellauer, M., Adler, M., Kinsy, M., Parashar, A. and Emer, J.: HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing, *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, pp. 406–417 (online), DOI: 10.1109/HPCA.2011.5749747 (2011).

[7] Schelle, G., Collins, J., Schuchman, E., Wang, P., Zou, X., Chinya, G., Plate, R., Mattner, T., Olbrich, F., Hammarlund, P., Singhal, R., Brayton, J., Steibl, S. and Wang, H.: Intel Nehalem Processor Core Made FPGA Synthesizable, *Proceedings of*

the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '10, New York, NY, USA, ACM, pp. 3–12 (online), DOI: 10.1145/1723112.1723116 (2010).

[8] Tan, Z., Waterman, A., Avizienis, R., Lee, Y., Cook, H., Patterson, D. and Asanović, K.: RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors, *Proceedings of the 47th Design Automation Conference*, DAC '10, New York, NY, USA, ACM, pp. 463–468 (online), DOI: 10.1145/1837274.1837390 (2010).

[9] Tan, Z., Waterman, A., Cook, H., Bird, S., Asanović, K. and Patterson, D.: A Case for FAME: FPGA Architecture Model Execution, *SIGARCH Comput. Archit. News*, Vol. 38, No. 3, pp. 290–301 (online), DOI: 10.1145/1816038.1815999 (2010).

[10] Wang, D., Jerger, N. and Steffan, J.: DART: A programmable architecture for NoC simulation on FPGAs, *Networks on Chip (NoCS), 2011 Fifth IEEE/ACM International Symposium on*, pp. 145–152 (2011).

[11] Wolkotte, P., Hölzenspies, P. and Smit, G.: Fast, Accurate and Detailed NoC Simulations, *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, pp. 323–332 (online), DOI: 10.1109/NOCS.2007.18 (2007).

[12] Wolkotte, P., Hölzenspies, P. and Smit, G.: Using an FPGA for Fast Bit Accurate SoC Simulation, *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–8 (online), DOI: 10.1109/IPDPS.2007.370374 (2007).

[13] Xilinx Inc.: Virtex-6 FPGA ML605 Evaluation Kit, http://www.xilinx.com/products/boards-and-kits/EK-V6-ML605-G.htm.