Vol.2011-ARC-194 No.13
2011/3/11

Distributed Shared-buffer ルータの遅延を削減するパイプラインバイパス方式

姜　軒[†1]　佐藤真平[†1]　吉瀬謙二[†1]

近年プロセッサアーキテクチャはコア数の増加の一途を辿っている．コア数の増加に伴い，コア間の通信遅延，スループットがアプリケーションに与える影響が益々大きくなってきている．コア間の通信には Network-on-Chip(NoC) が広く用いられるため，低通信遅延，ハイスループットオンチップネットワークルータの開発が急務である．提案されている DSB(DistributedShared-Buffer) オンチップルータがハイスループットであるが，パイプラインステージ数が多いため，通信遅延が大きい．本稿では，DSB オンチップルータの遅延削減を目的として，パイプラインをバイパスする方式を提案する．そして，サイクルアキュレート・フリットレベルシミュレータを使って，本方式の有用性を示す．

Pipeline Bypassing for Reducing Delay of Distributed Shared-Buffer Router

Jiang Xuan,[†1] Shimpei Sato[†1] and Kenji Kise [†1]

The number of cores in a processor is increasing these years. This will limit the application performance due to throughput and delay of network. Network-on-Chip(NoC) architectures are becoming defacto fabric for the multi-processors now. High throughput and low latency router are important for performance of many-core processor. DSB(DistributedShared-Buffer) on-chip router is high throughtput, but it is also high latency because of long pipeline. In this paper, we propose an efficient pipeline design using pipeline bypassing for DSB on-chip router in order to reduce the delay. By using the cycle-acculate flit-level simulator, we prove that it can actually reduce delay.

†1 東京工業大学 大学院情報理工学研究科
Graduate School of Information Science and Engineering, Tokyo Institute of Technology

1. はじめに

広い分野にわたり，複数のコアを搭載するマルチコアプロセッサ（マルチコア）が主流となりつつある．半導体の集積度の向上により，搭載されるコア数は着実に増加し続けており，プロセッサアーキテクチャはさらに多くのコアを集積するメニーコアプロセッサ（メニーコア）[*1] へと向かっている．しかし，コア数の増加に伴い，ハイスループット，遅延がアプリケーションに与える影響がますます大きくなる．コア間の通信に NoC が広く使われているため，ハイスループット，低遅延の NoC ルータの開発が今後ますます重要になると考えられる．

現在インプットバッファルータが典型的なオンチップルータとして広く利用されるが，アウトプットバッファよりスループットが低いため，インプットバッファルータのスループットを高めることは重要な課題である．ハイスループットであるルータとして，DSB(Distributed Shared-Buffer) オン，チップルータが提案されている．しかし，DSB ルータではメモリとクロスバースイッチを搭載したことと制御方法が複雑になったことにより，パイプラインステージ数の増加を招き，通信遅延が大きくなった．しかも，それらの問題はコア数の増加に伴い更に深刻になり，メニーコアプロセッサへの適用を困難にする見通しである．

そこで本論文では，DSB ルータの遅延削減を目的として，パイプラインパイパス方式を提案する．DSB オンチップルータにいて，入力バッファとメモリ間のクロスバーとメモリを使わず，バイパスルートを利用して，フリットを転送する方法を提案する．提案手法により，DSB のパイプラインステージ段数を 5 から 4 に減らせる．本論文ではネットワークオンチップのフリットサイクルレベルシミュレータを用いた評価により，DSB オンチップルータの遅延を削減できることを明らかにする．

以下，2 章では，インプットバッファルータについて説明する．3 章では，DSB について説明する．4 章では，インプットバッファルータと DSB ルータを比較して，提案手法を述べる．5 章では，実装方法について，説明する．6 章では，評価を行い，提案手法の有用性を示す．6 章では，関連研究を紹介する．7 章では，結論，今後の課題を述べる．

Vol.2011-ARC-194 No.13
2011/3/11

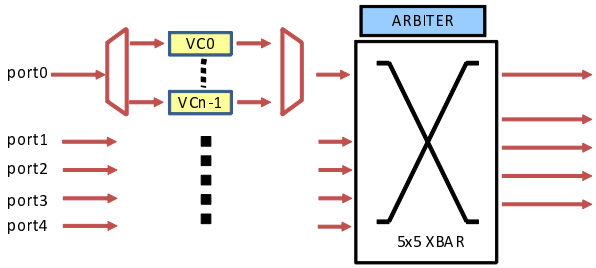

図 1 インプットバッファルータアーキテクチャ

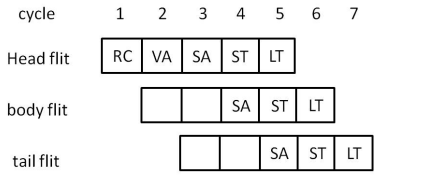

図 2 5 パイプラインステージインプットバッファルータのパイプライン構成

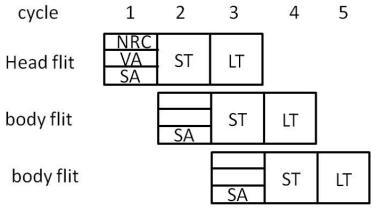

図 3 3 パイプラインステージインプットバッファルータのパイプライン構成

2. インプットバッファルータ

典型的なオンチップルータとして，インプットバッファルータ[1] が広くに使用されている．本節では，インプットバッファルータについて説明する．

2.1 virtual channel(VC) ルータ

図 1 に virtual channel 制御プロトコル[2] を採用した典型的な Wormhole ルータのマイクロアーキテクチャ[1] を示す．このルータはインプットバッファ，クロスバースイッチ，arbiter 回路から構成されており，各入力ポートに 2 本の仮想チャンネルを持つ．

図 2 にパイプライン構成を示す．フリットは入力線を経由して，割り当てられたバッファに格納される．ヘッドフリットは RC(Routing computation) ステージで，フリットに格納されている宛先アドレスにより出力ポートの計算をされ，VA(virtual channel allocation) ステージで，出力チャンネルに割り当てられる．フリットは SA(crossbar switch allocation) ステージで，クロスバースイッチ調停を行われ，ST(Switch traversal) ステージでクロスバースイッチを通過する．最後にフリットは LT(Link Traversal) ステージで，隣接ルータへ転送される．このルータは RC，VA，SA，ST，LT がそれぞれ 1 サイクル/ステージかかる，5 サイクル/ホップタイプである．

2.2 look-ahead ルータ

look-ahead ルータでは，RC ステージで次ホップのルータの出力ポートを計算する．VA ステージでは，ルータは前ホップのルータ/ローカル interface にてパケットに格納された出力ポート番号を用いて，パケットをフリーな出力チャンネルに割り当てる．以後の処理は通常のルータと同じである．look-ahead ルータでは，RC と VA 処理のデータ依存関係がなくなり，RC と VA を並列に処理する．look-ahead ルータは通常のルータより 1 パイプラインステージが少ない．

2.3 Speculative ルータ

Speculative ルータでは，非投機的な SA，投機的な SA と VA を並列に処理する．Speculative ルータは投機的な SA arbiter を用いて，入力チャンネルの割り当てを行う．そして，VA 処理が成功するなら割り当てられている入力チャンネルリストを更新する．但し，投機的な SA と VA が共に成功するなら，投機的な SA で選択した入力チャンネルが有効になる．また，投機的な SA で選択した入力チャンネルと非投機的な SA で選択した入力チャンネルが同時に有効である場合，非投機的な SA で選択したチャンネルを優先的に使う．Speculative ルータは VA と SA を並列に行うため，通常のルータより 1 パイプラインステージが少ない．

2.4 3 パイプラインステージインプットバッファルータ

この節では，これまでに述べた技術を採用した典型的なインプットバッファルータのパイプライン構成を説明する．

図 3 に典型的なインプットバッファルータのパイプライン構成を示す．第一ステージで，次ホップのルータの RC，VA と SA を行い，第二ステージで ST を行い，最後のステージで，LT を行う．このルータはフリットを 3 サイクルで転送する．

3. DSB(Distributed Shared-Buffer) ルータ

ハイスループットルータとして，DSB(Distributed Shared-Buffer) NoC ルータが提案さ

⋆1 1 チップに 2 個以上のコアを搭載するプロセッサをマルチコアプロセッサと呼ぶことにする．その中で，特に，1 チップに数十個以上のコアを搭載するプロセッサをメニーコアプロセッサと呼ぶことにする．

Vol.2011-ARC-194 No.13
2011/3/11

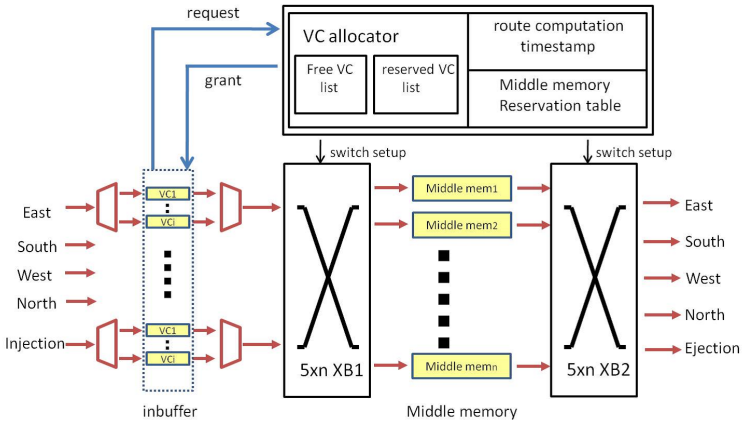

図 4 DSB ルータアーキテクチャ

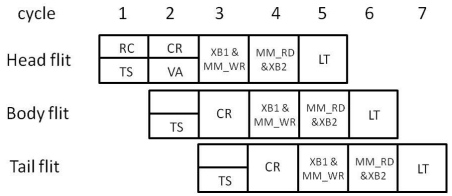

図 5 DSB ルータパイプライン構造

れている．本節では，DSB NoC ルータ[3] のアーキテクチャを説明する．

3.1 DSB ルータアーキテクチャ

図 4 に DSB ルータのアーキテクチャ図を示す．このルータは入力バッファ，2 つのクロスバ，n 本の 1 出力ポート 1 入力ポートの中間メモリ，arbiter 回路から構成されており，各入力ポートに i 本の仮想チャンネルを持つ．

このルータは 2 つのクロスバースイッチと中間メモリを使って，通常のルータの出力バッファを共有化するタイプである．フリットは入力配線を経由して，入力バッファに格納され，以下の処理をされる．

RC(route computation)

DSB ルータは look-ahead を採用したため，パケットの次のポップの出力ポート番号を計算する．通常の look-ahead ルータと同じである．(節 2.2)

TS(timestamping)

フリットは出力ポートを通過し，隣接ルータへ転送される timestamp(予定時刻) が計算される．(節 3.2)

CR(comflict resolution)

ルータは中間のメモリに読み込まれる/書き込まれる時の衝突を回避するため，フリットを適切な中間メモリに割り当てる．(節 3.3)

VA(virtual channel allocation)

フリットを仮想出力チャンネルに割り当てる．

XB1+MM_WR(via XB1 and write memory)

フリットはクロスバースイッチ 1 を経由し，中間メモリに格納される．

MM_RD+XB2(read memory and via XB2)

フリットは中間メモリから読み出され，クロスバースイッチ 2 を通過し，出力ポートのレジスタに格納される．

LT(link traversal)

フリットは出力ポートのレジスタから次のホップのルータに転送される．

図 5 に，DSB ルータのパイプライン構造を示す．RC と TS を並列に第一ステージで行い，CR と VA を並列に第二ステージで行い，XB1+MM_WR を第三ステージで行い，MM_RD+XB2 を第四ステージで行い，LT を第五ステージで行う．このルータは 5 段パイプラインタイプのルータであり，フリットが入力されてから次のホップのルータへ転送されるまで，最低 5 サイクル/ホップを要する．

3.2 TS(timestamping) ステージ

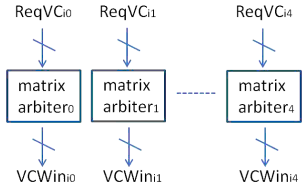

図 6 DSB ルータ TS ステージの Arbitation

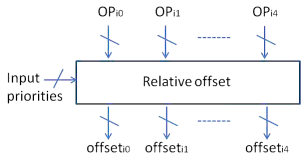

図 7 DSB ルータ TS ステージのオフセット計算

TS ステージでは，フリットが MM_RD+XB2 ステージで中間メモリから読み出される

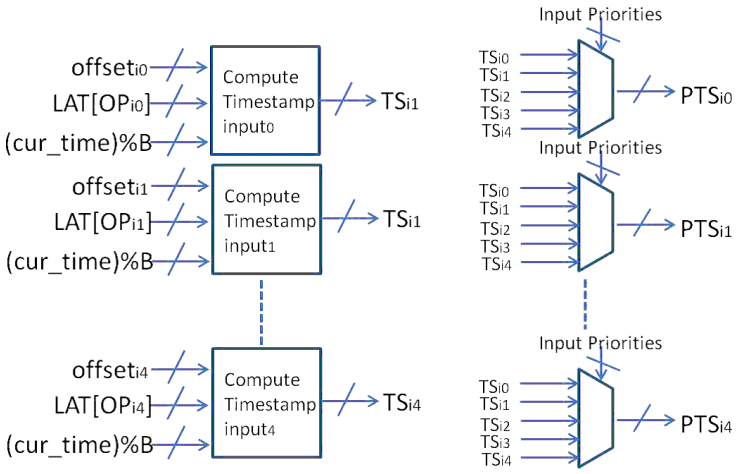

図 8 DSB ルータ TS ステージの Timestamp 計算

タイミングが計算され，次のポップのルータへ転送される予定時刻が決定される．これを実現するために，ルータは各フリットが出力ポートを通過するタイミングをユニークにさせ，出力ポート競合を回避するためのクロスバースイッチ調停を行う．TS 処理の詳細を以下に示す．

- timestamper は入力チャンネルの先頭フリットの予定時刻を計算する．先頭のフリットが第 2 パイプラインステージに入ったらに，入力チャンネルの 2 番目のフリットの予定時刻を計算する．
- パイプライン第 2 ステージにおいて，VA と CR(節 3.3 に説明する) が失敗になった場合には，先頭フリットが第 1 ステージに戻り，ST 処理をやり直す．
- 1 入力ポートの複数の入力チャンネルが timestamper に要求を出したときに，timestamper は LRU 方式でフリットを選ぶ．
- 1 サイクルに高々1 フリットが出力ポートを通過するため，同じ出力ポートを通過するフリットにユニークな timestamp がつく．

フリットの timestamp を計算する方法について述べる．要求を出したフリットの出力ポートを p として，timestamp を決める関数を T[p] とする．まず，中間メモリに出力 p のフリットが格納されていない場合の，T[p] の計算式を (1) 式に示す．

$$T[p] = Current\_Time + 3 \quad (1)$$

フリットが出力ポートに着くまでには 3 パイプラインステージ（サイクル）がかかるため現在の時刻に 3 を加える．

次に，中間メモリに出力 p のフリットが格納されている場合の計算方法を述べる．timestamper は毎サイクルに各出力ポートの timestamp を記録する．前サイクルの出力ポート p の最大 timestamp を関数 LAT[p] とする．他のフリットには LAT[p] と同じか小さい timestamp がついているから，出力の衝突を回避するために，フリットに LAT[p] より大きい timestamp をつける．したがって，timestamp の計算は (2) 式に示す通りである．

$$Timestamp = max(LAT[p] + 1, T[p]) \quad (2)$$

また，複数のフリットが同じポートに要求を出したときに，timestamper が優先度が高いフリットから順に小さい timestamp をつける．この時の timestamp の計算方法を以下に説明する．但し，中間メモリの出力 p のフリットは格納されていないとする．**図 6** に示すように，matrix arbiter を用いて，各ポートの複数入力チャンネルからひとつを選び出す．各入力ポートから出力されるフリットは同サイクルに同じ出力ポートを通過できない．そこで，フリットの優先度により，フリットのオフセットが計算され，timestamp が計算される．**図 7** に示すように，入力ポート優先度（OP）により，フリットのオフセットが計算される．一番優先度が高い入力ポートのフリットのオフセットが 0 になり，残りの同じ出力ポートのフリットを持つオフセット（offset）が優先度の順次にインクリメントされる．このときの timestamp は (3) 式に示す通りである．

$$Timestamp = Current\_Time + 3 + offset_i \quad (3)$$

これらをまとめて一般化すると，timestamp 値の計算方法は (4) 式を示す通りである．

$$TS_i = max(LAT[OP_i] + 1, current\_time + 3) + offset_i \quad (4)$$

timestamp を計算するブロック図は**図 8** に示すようになる．各入力ポートからのフリットが対象とする出力ポートの前サイクルにおける timestamp の関数 $LAT[OP_i]$，フリットのオフセット $offset_i$，現在時刻 $Current\_Time$ が timestamper ブロックに入力される．オフセットを加算された timestamp 値が timestamper ブロックから出力される．

TS_i は入力ポートの優先度により並べられ，PTS_i となる．これは CR ステージで用いられる．

3.3 CR(comflict resolution) ステージ

DSB ルータに搭載されている中間メモリは，1 出力ポート 1 入力ポートであるため複数のフリットを中間メモリに書きこむ/読み出すときに競合が起きる．CR ステージでは，こ

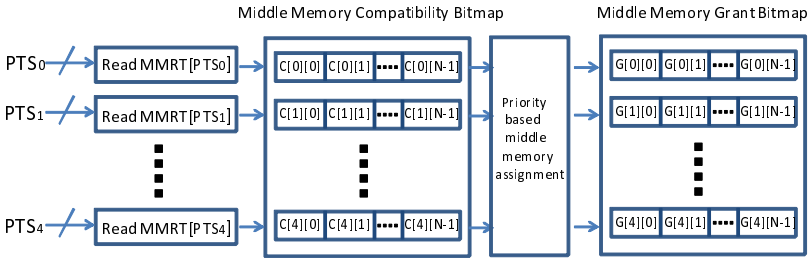

図 9 DSB ルータの CR ステージの構成

うした競合が起こらないように，フリットを書き込む中間メモリを決める．**図 9** に CR ステージのブロック図を示す．

まずに，中間メモリから読み出す時の衝突を回避する方法を述べる．各中間メモリにユニークな timestamp を持つフリットを格納することで，中間メモリから読み出すときの衝突を回避できる．DSB ルータは Middle Memory Reservation Table（MMRT）と呼ばれる，ある timestamp にどの中間メモリが読みだされるかを格納するテーブルをもつ．このテーブルを PTS_i で引くことにより，**図** 9 に示す Middle Memory Compatibility Bitmap（MMCB）を構成する．このとき，C[i][j] は timestampPTS_i における中間メモリ j の使用状況を示す．よって，C[i][j] が 0 である中間メモリを選ぶことで，複数フリットが 1 つの中間メモリから読みだされることを回避できる．

次に，中間メモリに書きこむ時の衝突を回避する方法を述べる．フリットを異なる中間メモリに書き込むことで，中間メモリに書きこむときの衝突を回避できる．**図** 9 に示される Middle Memory Grant Bitmap （MMGB）の G[i][0] から G[i][N-1] までの N ビットのレジスタが中間メモリに書き込む時に利用される．G[i][j] のビットが立つとき，中間メモリ j は優先度 i の入力ポートからのフリットが書き込まれることを意味する．より優先度の低い入力ポートが中間メモリ j に書き込むことは禁止される．よって，複数フリットを同時に 1 つの中間メモリに書き込むことを回避できる．

4. パイプラインをパイパスする DSB ルータ

本章では，まず 3 パイプラインステージインプットバッファルータと DSB ルータを考察し，DSB ルータの課題を明らかにする．次にパイプラインをバイパスする DSB ルータを提案する．

表 1 チャンネル数とバッファサイズ

Config.	Input buffers(per port)		Middle memmory buffers		Total buffers
	#VCs	Flits/VC	MM	Flits/MM	
Input-buffered router	8	5	-	-	200
DSB router	5	4	5	20	200

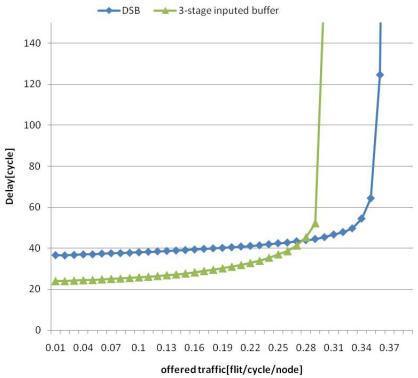

図 10 インプットバッファルータと DSB ルータの比較

4.1 インプットバッファルータと DSB ルータの考察

節 2.4 で述べたインプットバッファルータは 3 段パイプラインタイプのルータである．一方，節 3 で述べた DSB ルータはインプットバッファルータよりハイスループットであるが，構成と制御が複雑であり，5 段パイプラインタイプのルータである．

図 10 に，8×8 の 2D メッシュ，ランダム通信において，インプットバッファルータと DSB ルータの性能比較図を示す．縦軸は遅延である．横軸は offered traffic である．表 1 にルータのチャンネル数とバッファサイズを示す．インプットバッファは各入力ポートに 8 本の仮想チャンネルを持ち，各チャンネルのバッファサイズが 5 フリット分なので，トータルバッファサイズは 200 フリット分である．DSB ルータは各入力ポートに 5 本の仮想チャンネルを持ち，各チャンネルのバッファサイズは 4 である．また，DSB ルータは 5 本の 1 出力ポート 1 入力ポートの中間メモリを持ち，各メモリのサイズが 20 フリット分である．トータルバッファサイズはインプットバッファルータと同じ 200 フリット分である．図 10 に示すように，ネットワークのスループットが小さいときに DSB ルータの遅延がインプットバッファルータよりかなり大きい．それは，DSB ルータのパイプライン段数が多く，パ

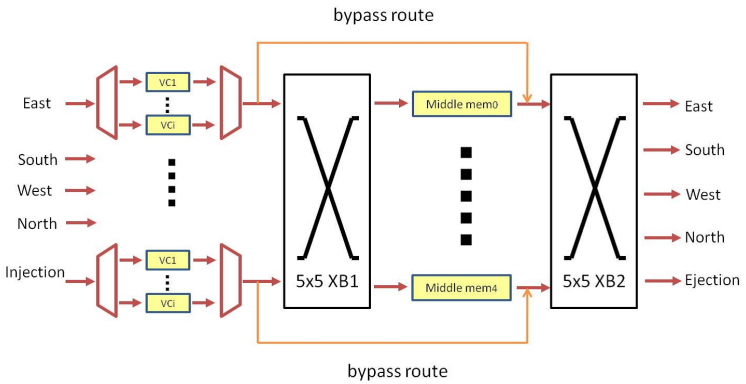

図 11 提案手法のアーキテクチャ

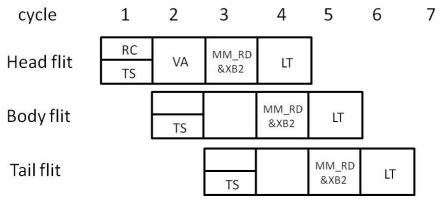

図 12 バイパスが成功した場合のパイプライン構造

ケットを転送するサイクル数が大きいと考えられる．よって，DSB ルータのパイプライン段数を減らす必要がある．

4.2 パイプラインをパイパスする DSB ルータの提案

節 4.1 に述べたように，DSB ルータはハイスループットであるがパイプライン段数が多いため通信遅延が増加してしまった．

ルータのパイプライン段数は通信遅延に大きく影響するため，ルータのパイプライン段数を減らすバイパス制御を提案する文献がいくつか挙げられる[4)5)]．それらの研究では，パイプラインをバイパスする制御を用いて，コア間の通信遅延を削減することが明らかにされている．

そこで，本研究はパイプライン段数の多い DSB ルータを低遅延化するため，パイプラインをバイパスする制御方法を提案する．このパイプラインをバイパスする DSB ルータには，従来の DSB の第 3 パイプラインステージ (XB1+MM_WR) をスキップする制御を加える．バイパス条件を満たす場合に，フリットを 4 サイクルで転送できる．バイパス条件を満たさない場合には従来手法と同様にフリットを 5 サイクルで転送する．すなわち，バイパス制御に関するペナルティがない．同じタイミングで同じ出力にバイパスできるフリットが複数存在する場合は，優先度が一番高いフリットのみをバイパスルート配線で転送する．

図 11 に DSB ルータにおいて，バイパスルートを挿入したアーキテクチャ図を示す．フリットがバイパスされる際の処理について説明する．フリットは入力バッファからディマルチプレクサを通過し，中間メモリの右のクロスバを経由し，隣接ルータへ転送される．この場合，クロスバ 1 と中間メモリを使わず，フリットを転送するため，従来の第 3 パイプラインステージ (XB1+MM_WR) をスキップできるようになる．TS 処理では，バイパスされるフリットの予定時刻は通常フリットより 1 サイクル早い時刻に決定される．

図 12 にバイパスが成功した場合のパイプラインステージ構成を示す．パイプラインステージ 1 にバイパスを判断する処理 (BC) を追加する．そこで，次のサイクルにバイパスできるなら，次に入力されたフリットがパイプラインステージ 3 に入らず，バイパスルートを経由し，第 2 のクロスバースイッチに入る．フリットが入力されてから次のホップのルータへ転送されるまで，4 サイクルを要する．5 章で，その実装方法について述べる．

5. パイプラインをパイパスする DSB ルータの実装

5.1 バイパスの制御方法

毎サイクルにバイパスチェック (BC) を行い，次のサイクルにパイパス条件を満たすならバイパスレジスタに 1 を書き込む．バイパス条件を満たさないならバイパスレジスタに 0 を書き込む．TS 処理では，バイパスレジスタを参照し，フリットにバイパスの timestamp をつける．バイパスの timestamp がついているフリットがバイパスルートを通過する．

5.2 バイパスされるフリットの timestamp 処理

バイパスされるフリットの TS 処理では，バイパスレジスタを参照し，バイパスされるフリットに timestamp をつける．但し，バイパスされるフリットは TS ステージから TL ステージまで 2 サイクルかかるから，TS の計算式は式 1 ではなく，式 $T[p] = Current\_Time + 2$ になる．バイパスするフリットは中間メモリに格納されないため，CR ステージでは，バイパスされるフリットに対する処理を行わない．VA では，従来手法と同じように，バイパスされるフリット (ヘッドフリット) をフリーな出力チャンネルに割り当て，失敗する場合に

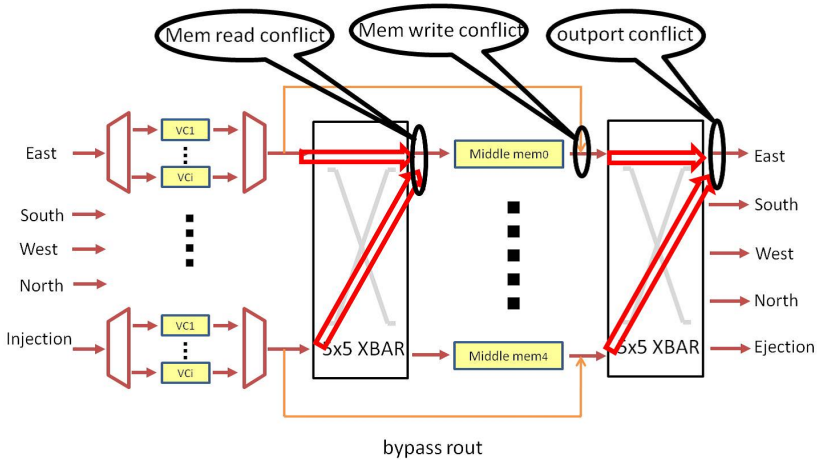

図 13 DSB ルータにおける三種類の衝突

フリットを TS ステージに戻す．

5.3 中間メモリに読み込む時の衝突を回避する方法

節 3.2 と節 3.3 に述べたように DSB ルータにおいては，出力ポートへの出力，中間メモリからの読み出し，中間メモリへの書き込みの 3 種類の衝突がある．

図 13 に 3 種類の衝突を示す．出力ポートに起きる衝突の回避方法は従来手法と同様，同じ出力ポートを通過するフリットにユニークな timestamp をつけることである．また，バイパスされるフリットはメモリに書き込まれないため，通常のフリットと衝突が起きない．したがって，中間メモリに書き込む時に起きる衝突の回避方法も従来手法と同様，CR ステージで MMGB を使用し，同時に複数のフリットを同じメモリに書き込むことを禁止することである．しかし，中間メモリから読みだす時に起きる衝突は従来の手法では回避できない．衝突が起きる原因と回避方法を以下に説明する．

パイプラインをパイパスする DSB ルータは各入力バッファの右のディマルチプレクサの出力配線から相対する中間メモリの出力配線にバイパスルート配線が接続されている．同じタイミングでバイパスルートと中間メモリからフリットが流れればなら衝突が起きる．そのため，バイパスルートを通過したフリットに対して中間メモリに格納されているフリットと異なる timestamp を持たせる．節 3.3 に述べたように，従来手法では MMCB を利用して，同じ中間メモリにユニークな timestamp を持つフリットを格納することで．中間メモリから読みだす時の衝突を回避する．これと同様に，提案手法では，BC ステージで，MMCB を用いて，中間メモリに $Current\_Time+3$(次のサイクルにバイパスするフリットの)timestamp を持つフリットが格納されていないなら，次のサイクルにフリットをバイパスさせる．しかし，CR 処理後のフリットの timestamp が MMCB に格納されるため，パイプラインステージ 1 とパイプラインステージ 2 の間のパイプラインレジスタに格納されるフリットが後でバイパスと同じ中間メモリに書き込まれるなら，バイパスするフリットと衝突する．また，パイプラインレジスタに格納されるフリットは CR ステージ前に書き込まれる中間メモリの位置を確認できない．そのためパイプラインレジスタの中にフリットの timestamp が $Current\_Time+3$ であるものが 1 つでもあるならバイパスを行わない．

サイクルのフリットがバイパスできる条件を以下にまとめる．

(1) 相対する中間メモリに timestamp が $Current\_Time+3$ であるフリットが格納されていない

(2) パイプラインステージ 1,2 の間のパイプラインレジスタに timestamp が $Current\_Time+3$ であるフリットが格納されていない

BC ステージで以上の条件を判断する処理を行う．

5.4 パイプラインをパイパスする DSB ルータの実現性

以上に示した通りにパイプラインをバイパスする DSB ルータは DSB の上にバイパスするルートと BC 処理を追加したものである．バイパスルートの挿入はハードウェアで実現できることが明らかである．BC ステージ処理の実現性について述べる．

節 3.3 に，CR ステージで中間メモリを読み出す時の競合を回避する方法を述べた．CR ステージで MMRC から MMCB を構成し，中間メモリに格納されているフリットの timestamp を確認する処理を行う．BC ステージにおける条件 1 の判定は，この CR 処理とよく似ている．実際には，MMRT の読み出しポート数を 1 つ増やし，MMRT[$Current\_Time+3$] をチェックすることで実現できる．また，条件 2 の判定は，第 1，2 パイプライン間のパイプラインレジスタに格納されているフリットの timestamp を比較する回路を追加することで実現できる．したがって，BC ステージ処理の実現性はあると言える．

以上により，パイプラインをパイパスする DSB ルータには実現性があると考えられる．

6. 評　　価

本章では，フリット · サイクルレベルのネットワークシミュレータを用いて，パイプラインをパイパスする DSB ルータの評価を行う．

Vol.2011-ARC-194 No.13
2011/3/11

6.1 パイプラインをバイパスする DSB ルータの性能評価

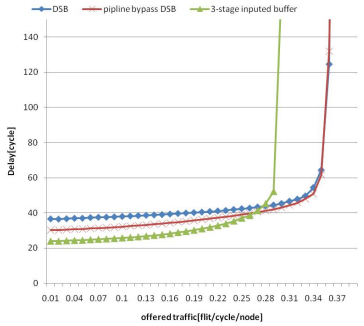

図 14 8×8 のネットワークのランダム通信

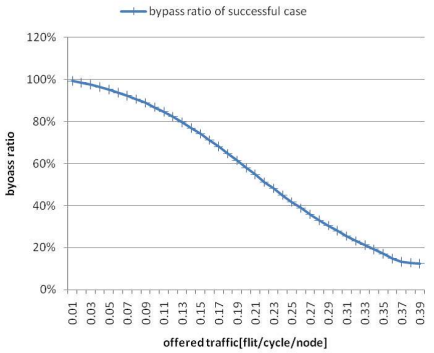

図 15 8×8 のネットワークのランダム通信のバイパス成功率

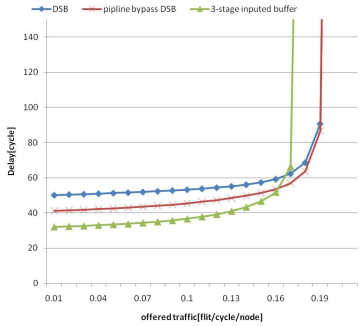

図 16 8×8 のネットワークのビットコンプリメント

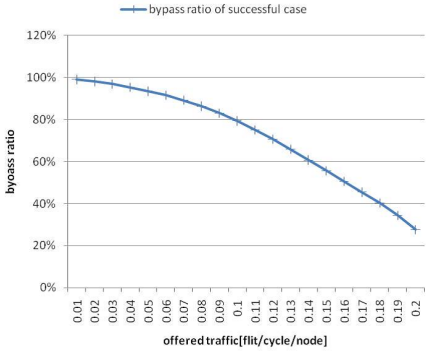

図 17 8×8 のネットワークのビットコンプリメントのバイパス成功率

ウォームアップフェースのサイクル数を 5000 に設定し，測定フェースのサイクル数を 100,000 に設定する．ルーティングアルゴリズムとして，X-Y 次元順ルーティングを用いる．パケットサイズを固定の 4 フリットに設定する．シミュレーションに用いる通信トラフィックパターンとして，ランダム通信，ビットコンプリメントとトルネードを用いる．測定するネットワークサイズは 8×8 とする．トラフィックレートを変化させて，遅延サイクル数とパイプラインがバイパスされた割合を計測する．

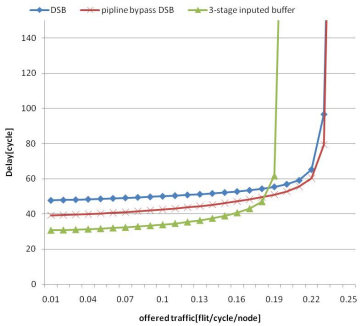

図 18 8×8 のネットワークのトルネード

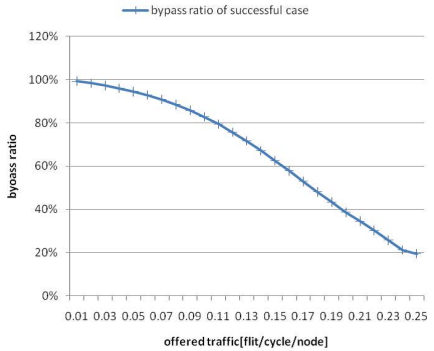

図 19 8×8 のネットワークのトルネードバイパス成功率

textgt **図 14**，**図 16**，**図 18** に遅延に関する評価の結果を示す．縦軸はパケットの遅延，横軸は指定したトラフィックレートである．青い線が通常の DSB ルータ，赤い線がパイプラインをバイパスする DSB ルータを表す．また，**図 15**，**図 17**，**図 19** にパイプラインがバイパスされた割合の評価結果を示す．縦軸はバイパスの成功率である．トラフィックが低い時はバイパスの成功率が高く，遅延の削減率が一番高い．トラフィックが高くなるとほとんどバイパスできなくなり，パイプラインをバイパスする DSB ルータが通常の DSB ルータと同じ通信遅延を持つ．また，トラフィックが低い時にパイプラインをバイパスする DSB ルータの遅延が通常の DSB ルータと 3 パイプラインステージインプットバッファルータの中間であり，予想通りの 4 パイプラインステージ動作になる．

表 2 にパイプラインをバイパスする DSB ルータによる遅延の削減率の結果を示す．最大の最大遅延の削減がおよそ 17.9%，平均の最大遅延の削減率がおよそ 17.6%を達成している．また，**図** 14，**図** 16，**図** 18 に示すように，バイパスによるスループットへの悪影響は見られなかった．

表 2 ネットワーク性能評価の結果

通信トラフィックパターン	最大遅延の削減 (%)
ランダム通信	17.1
ビットコンプリメント	17.9
トルネード	17.8

7. 関連研究

提案するパイプラインをバイパスする DSB ルータは，パイプライン段数が多い DSB を対象とし，パイプラインをバイパスする制御を追加することで，DSB ルータのパイプライン段数を減らす．パイプライン段数の減少によって，ネットワークの遅延を削減するアプローチである．本章では，関連する研究について述べる．

7.1 オンチップルータ

加速の仮想チャンネルによるバイパス制御が提案されている[4]．フリットは次 2 ホップのルータまでの方向が決定できるなら，次の 2 ホップのルータに特定の仮想チャンネルを使い，VA/SA ステージをバイパスする方式である．しかし，フリットが経由する 3 つのルータにおいて，最大 2 回までバイパスしないことがこの方式の欠点である．提案手法のバイパス方式ならば，出力ポートの衝突がなければすべてのルータでパイプラインをバイパスできる．

Token flow control というバイパスフローコントロールが提案されている[5]．フリットが次のホップのルータに到着する前に，次ホップのルータからの制御信号に基づき，バイパス制御を行う．条件を満たした場合に次のホップのルータはフリットを 1 サイクルで転送する．制御信号が隣接ルータのリソースより決まる．Token flow control は広くネットワークのリソースをチェックするバイパス方式だが提案手法のバイパス方式は自分のルータをしかチェックしない．

予測機能を持つルータが提案されている[6]．入力されたフリットの方向を予測し，予測が当たれば 1 サイクルでフリットを転送し，予測が外れれば通常通りフリットを転送する．DSB ルータのバイパス方式との一番違いはバイパス条件にある．予測ルータは予測アルゴリズムにより，バイパス成功率が変わる．一方，提案手法のバイパス方式は中間メモリ読み出す衝突より，バイパス成功率が変わる．

また，次元順ルーティングの規則性を利用したバイパス方式[7]，頻繁に使われるパス上の転送を低遅延化にするバイパス方式[8]，ユーザ指定の特定パス上の転送を低遅延化にするバイパス方式[9] は提案されている．それらのバイパス方式は転送ルートまたはルーティング方法に依存しているが提案手法のバイパス方式は転送ルートまたはルーティング方法に依存しない．

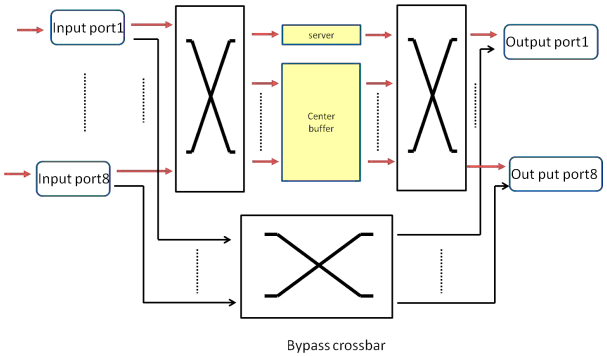

図 20 IBM Colony ルータの構成

7.2 オフチップルータ

IBM の Colony ルータが共有メモリタイプであり，そしてバイパス制御がついている．[10] 図 20 に IBM の Colony ルータ構成を示す．このルータは 8 入力ポート 8 出力ポート，2 つのクロスバースイッチ，中間メモリとバイパス用のクロスバースイッチから構成される．そこで，DSB ルータのバイパス方式と違うところは，IBM の Colony ルータはバイパスするために，クロスバースイッチを使うことである．オンチップルータでは，厳しいハードウェアコスト制限があるため，IBM の Colony ルータのバイパス方式はオンチップルータに適用しないと考えられる．

8. まとめと今後の課題

プロセッサのコア数の増加に伴い，コア間の通信がアプリケーションに与える影響が益々大きくなる．低遅延とハイスループットのオンチップルータのネットワークオンチップに関するさまざまな研究が進んでいる．その中で，ハイスループットである DSB ルータが提案されている．DSB ルータはパイプライン段数が多いため，遅延が大きい．

そこで，本論文では DSB ルータの低遅延化を目指し，パイプラインをバイパスする制御を提案した．そして，フリット . . . サイクルレベルのシミュレータを用いて，ランダム通信，ビットコンプリメント，トルネードの通信パターンで，評価を行った．結果，8 × 8 ネット

ワークにおいて，およそ 17.1%の最大遅延の削減率を達成し，DSB ルータの低遅延化を明らかにした．また，ネットワーク規模の増大に伴い，提案手法による遅延の削減効果が上がることを明らかにした．

本研究における今後の課題について述べる．まず，アプリケーションのトレースを用いて，パイプラインをバイパスする DSB ルータを評価することを挙げられる．また，ハードウェア記述言語を使い，パイプラインをバイパスする DSB ルータの回路面積と消費電力を評価することが考えられる．そして，低遅延化を目指し，更にパイプラインステージをバイパスする手法がないか検討する必要がある．

参 考 文 献

1) Dally, W.J. and Towles, B.: *Route Packets, NotWires: OnChip Interconnection Networks*, Morgan Kaufmann (2004).
2) Dally, W.J.: Virtual-channel flow control, *Proceedings of the 17th annual international symposium on Computer Architecture*, ISCA '90, New York, NY, USA, ACM, pp.60–68 (1990).
3) Ramanujam, R. S., Soteriou, V., Lin, B. and Peh, L.-S.: Design of a High-Throughput Distributed Shared-Buffer NoC Router, *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '10, Washington, DC, USA, IEEE Computer Society, pp.69–78 (2010).
4) Kumar, A., Peh, L.-S., Kundu, P. and Jha, N.K.: Express virtual channels: towards the ideal interconnection fabric, *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, New York, NY, USA, ACM, pp. 150–161 (2007).
5) Kumar, A., Peh, L.-S. and Jha, N.K.: Token flow control, *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 41, Washington, DC, USA, IEEE Computer Society, pp.342–353 (2008).
6) Matsutani, H., Koibuchi, M., Amano, H. and Yoshinaga, T.: Prediction router: Yet another low latency on-chip router architecture, *High Performance Computer Architecture, 2009. HPCA 2009. IEEE 15th International Symposium on*, pp.367 –378 (2009).
7) Izu, C., Beivide, R. and Jesshope, C.: Mad-postman : A Look-ahead Message Propagation Method For Static Bidimensional Meshes, *Parallel and Distributed Processing, 1994. Proceedings. Second Euromicro Workshop on*, pp.117 –124 (1994).
8) Park, D., Das, R., Nicopoulos, C., Kim, J., Vijaykrishnan, N., Iyer, R. and Das, C.R.: Design of a Dynamic Priority-Based Fast Path Architecture for On-Chip Interconnects, *Proceedings of the 15th Annual IEEE Symposium on High-Performance Interconnects*, HOTI '07, Washington, DC, USA, IEEE Computer Society, pp.15–20 (2007).
9) Michelogiannakis, G., Pnevmatikatos, D. and Katevenis, M.: Approaching Ideal NoC Latency with Pre-Configured Routes, *Proceedings of the First International Symposium on Networks-on-Chip*, NOCS '07, Washington, DC, USA, IEEE Computer Society, pp.153–162 (2007).
10) Stunkel, C.B., Herring, J., Abali, B. and Sivaram, R.: A new switch chip for IBM RS/6000 SP systems, *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '99, New York, NY, USA, ACM (1999).