VLDP3アーキテクチャにおける レジスタ値の高速Forwarding手法の提案

田中 裕治[†] 入江 英嗣[†] 服部 直也[†] 坂井 修一[†] 田中 英彦[†]

概要

次世代アーキテクチャVLDP3は、多数の ALU を規則的に配置し、ALU 間でデータフローを直接実行する。ALU 間は低レイテンシでデータ転送できる仕組みを取り入れる。しかしレジスタを介したデータ転送はスイッチング・ロジックの複雑さなどの影響によりレイテンシの増大を引き起こす。本稿では、ALU とレジスタ間でのアクセス・レイテンシの増加を抑えるため、多数 ALU 間で Forwarding を行い、レジスタを介さないレジスタ値転送機構を提案する。多数 ALU で構成された実行機構において効率的な Forwarding 機構を提案し、シミュレータによる評価を行った。

Register Forwarding Mechanisms on VLDP3 Architecture

Yuji Tanaka[§] Hidetsugu Irie[§] Naoya Hattori[§]
Shuichi Sakai[§] Hidehiko Tanaka[§]

Abstract

The next-generation architecture VLDP3 has many ALUs in order, and directly executes dataflow with ALUs. VLDP3 transfers data among ALUs with low latency, but data communication with register unit increases latency because of the complexity of switching logic, and so on. In this paper, in order to decrease register access latency among ALUs and register unit, we proposed register forwarding mechanisms among many ALUs which don't access register unit, and evaluated them by simulation results.

1 はじめに

半導体技術の進歩によって半導体プロセスが微細化し、チップ上に多数の資源を使用することができるようになった。しかしプロセスの微細化に伴って配線遅延の影響を無視できなくなるなど、性能低下を招く新たな要因を考慮する必要が出てきた。そのため近年では、性能のトレードオフを考えた新しいプロセッサ・アーキテクチャとして、多数の ALU で構成された実行機構を持つ VLDP3 などが研究されている [6]。

多数 ALU を規則的に配置した ALU-Net 構造では、ALU-Net にデータフローを割り当てて直接実行する [6]。この方式を使うと ALU 間のデータ転送が低レイテンシで行えるが、レジスタからデータを読み書きするためのレイテンシが相対的に増大してしまう。従来手法を使ってレジスタを介さない Forwarding を ALU 間で行えば、レイテンシの増加が抑えられる。しかし任意の ALU 間で Forwarding を行おうとするとロジックが複雑化してしまい、レイテンシ削減は難しくなってしまう。そこで、多数 ALU で構成された実行機構に適した Forwarding 機構を考える必要が出てくる。

本稿では、VLDP3 の実行機構とレジスタ機構の概略を述べ、レジスタ依存によって生じるレイテンシを抑えるメカニズムを考えていく。従来手法の問題点をまとめ、VLDP3 のような多数演算器構成の実行機構において、多数の演算器間でレジスタ値を Forwarding するための手法を提案し、その機構をシミュレータに実装して評価する。

2 VLDP3の概要

本章では VLDP3 アーキテクチャの概要について実行機構を中心に述べる。

2.1 ALU-Net の構成とネットワーク

VLDP3 の実行機構は、複数の Node を規則的に並べ、各 Node 間を短い配線 (Local Wire) で結合した ALU-Net である。Node は演算処理を行う最小単位で、ALU(演算ユニット) と制御ロジックから成る。各 Node のロジックは小規模かつ単純であるため、ALU-Net を大規模化しても複雑化が抑えられる利点がある。

[†] 東京大学
[§] Graduate School of Information Science and Technology, University of Tokyo

ALU-Net のモデルを図 1 に示す。各 Node は、ALU と Operand Buffer、そして小規模な Issue Logic から成る。IB は多重実行されるため、Operand Buffer は小規模なリザベーション・ステーションの役割を果たす。Issue Logic は命令の発火を制御する。発火された命令から順に演算を行い、演算結果を Router を通して次の Node へと転送する。各 Node 間を結ぶ Local Wire は短い配線で、配線遅延の影響を受けにくい。なおかつ単純なロジックで制御されるので、低レイテンシのデータ転送を可能にする。次の Node を繋ぐ配線は図のように 3 本に限定したものを想定する。

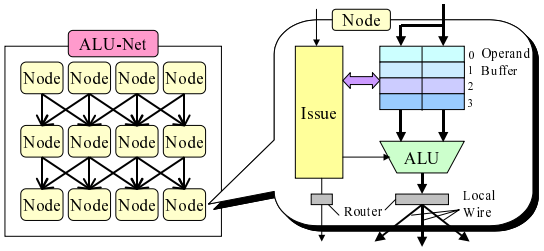

図 1: ALU-Net と Node の構成

ALU-Net 上にデータフローを形成する様子を図 2 に示す。命令 I1 と I2 はレジスタ R0〜R2 を読み込み、R6 へ書き出す。各 I1〜I4 の演算結果はそれぞれ Local Wire を使って受け渡しされるので、低レイテンシな転送が可能となる。一方の ALU-Net とレジスタ・ユニットを繋ぐネットワークの配線を Global Wire と呼ぶ (図 2(右) 太線)。

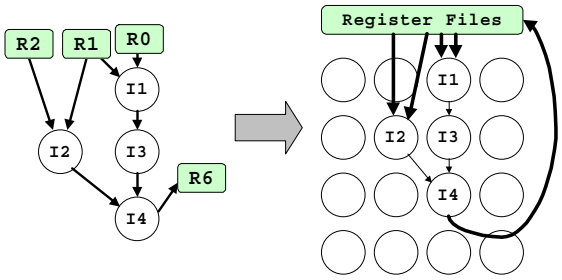

図 2: DFG(左) とアサイン後の ALU-Net(右)

2.2 IB 単位の処理方式

複数の命令列を静的に切り分け、IB (Instruction Block) という塊を構成する。命令を IB 単位にクラスタリングすることで、IB の中だけで受け渡しされるデータと、別の IB との間で受け渡しされるデータの 2 つのデータ転送に分類できる。できるだけ多くのデータ転送を IB 内だけに閉じ込められれば、IB 内と外とで別々のデータ転送手法を取り入れることで、局所性を利用した高速な転送が可能になる。

IB にはコンパイラによって IB 内データフローグラフ (DFG) の情報が付加される。この情報は、ALU-Net 内のどの Node へ命令を割り当てる (Assign) かという配置情報になっており、命令を割り当てられた Node ではこのデータフローグラフの情報にもとづいた演算を行う。また、処理を IB 単位にすることで、数十命令の単位でレジスタ・リネーミング処理を行うため、レジスタでの処理を単純化できる利点がある。

2.3 レジスタ管理ユニット

ALU-Net にレジスタ値を供給し、書き込み処理を管理するユニットをレジスタ管理ユニット (RWU: ReadWriteUnit) と呼ぶ。RWU では IB を単位として命令間の依存関係を解析し、レジスタ・リネーミング処理を行う。

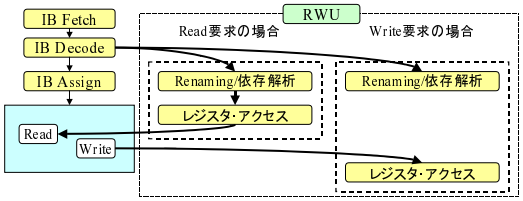

図 3: RWU へのアクセス要求

RWU での動作を図 3 に示す。レジスタ読み込み命令の場合は RWU 内でリネーミングと依存を解析して該当する Node へレジスタ値を転送する。レジスタ書き込み命令の場合にも同様に RWU 内でリネーミング処理と依存解析を行い、ALU-Net からレジスタ値を受け取る。RWU はレジスタアクセス要求と共に命令の座標情報を IB Decode ステージから受け取っているので、レジスタ値を必要な Node へ直接転送できる。

2.4 実行モデル

IB 実行のモデルを図 4 を用いて説明する。実行前処理は IB 単位で Fetch、Decode、Assign の順に実行される。レジスタ・アクセスはリネーム処理が終わると Global Network を経由して ALU-Net へ転送され、オペランドが揃った命令から順に発火する。演算結果は RWU へ出力される。IB 内の演算がすべて終わると ALU-Net 上から IB が flush される。命令が例外を起こさず、かつ直前の IB の Retire を確認すると IB Retire

を開始する。

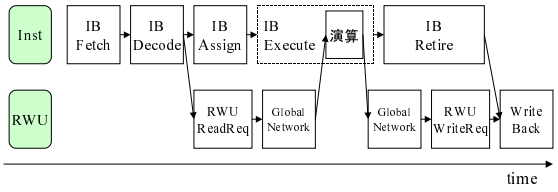

図 4: IB の実行モデル

3 データ依存の分類

ALU-Net 方式では、IB を境界としてデータの依存関係が 2 つ存在する。ただし本稿でデータと言った場合にはレジスタ値のことを指し、メモリについては関連研究 [8] で述べられている。ここで 2 つの依存関係をまとめる。

1. IB 内データ依存
   IB 内の命令間に真の依存関係がある場合で、これを IB 内データ依存と呼ぶ。IB 内データ依存の関係にある演算結果は Local Wire によって転送経路が静的に決まっておりレジスタを介さないため、次の Node へ高速に転送できる。
2. IB 間レジスタ依存
   IB を跨いだ命令間に真の依存関係がある場合を IB 間レジスタ依存と呼ぶ。この関係にある命令の間ではレジスタを介したデータ転送が必要なため、Global Wire を経由する。そのため、転送距離のアクセス・レイテンシが増大し、さらに IB を跨ぐレジスタ転送経路を RWU で動的に決定するためレイテンシが増大してしまう。

このことから、IB 間レジスタ依存にあるレジスタ値の転送がボトルネックになりやすく、高速化する必要がある。次章以降において、高速化のためにレジスタ値 Fowarding 機構の導入を考える。

4 VLDP3 での Forwarding

本章では従来の Forwarding 機構を VLDP3 に採用したときの問題点をまとめ、新たな機構の必要性を述べる。

4.1 従来手法の問題点

従来の Forwarding 手法では、演算結果を他の全ての ALU へ無条件にブロードキャストする必要があるため、ALU 数の増加とともに Result Bus からの入力数も増加し、Issue Logic が複雑化する問題を抱えていた。4way から 8way 程度のスーパースカラであれば、このような全 ALU へのブロードキャスト方式による Forwarding でも実現可能である。しかし、本研究で想定する VLDP3 アーキテクチャは、ALU-Net に多数の ALU を持っている。現時点で実行機構には 64 個の ALU を持った ALU-Net を想定しているため、もし仮に従来の Forwarding をそのまま適用したとすると、Result Bus からの入力数が 1Node あたり 128 入力 (=2 オペランド ×64ALU) になってしまう。しかも各 Node には Local Wire からの 6 入力 (=2 オペランド ×3 方向からの入力) とレジスタ・ユニットからの 2 入力があるため、実際の入力数はこれ以上となる。入力数が増えると、リザベーション・ステーションでの処理が複雑化する。そして Issue 処理のレイテンシが増大してしまう。さらに、大規模化による配線遅延の影響が無視できなくなることも予想される。

次章では、VLDP3 で Forwarding を実現するために、Issue Logic の複雑化を抑え、多数演算器で構成された実行機構の下での Forwarding 手法の確立を目指す。

4.2 Forwarding 対象依存距離

本稿では依存距離という言葉をレジスタ依存のある IB 同士が時間的にどれだけ離れているかを示す値と定義し、IB のフェッチ順の差で表されるものとする。IB 間レジスタ依存の 50 %以上は、依存距離 1 の IB 間 (=隣接 IB 間) であることがわかっている [4]。よって本研究ではこの局所性を生かし、隣接 IB のみを対象とした Forwarding 機構を提案する。

5 提案機構

5.1 提案機構の概要

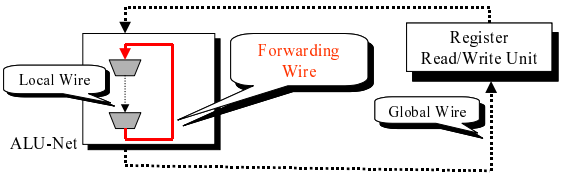

図 5: Wire の種類

提案機構の特徴は、IB 間のレジスタ依存解析を IB Assign 完了前に行い、依存関係の情報を ALU-Net 内の Node に持たせることにある。VLDP3 のベースラインモデル [6] ではレジスタ・アクセスの情報をすべて RWU 内で処理していたため、通信レイテンシが大きいという欠点を抱えていた。本方式では、IB 間のレジスタ依存関係を前

倒しで行い、その結果情報を ALU-Net 内にも持たせる。ALU-Net 内で依存情報を分散管理することで RWU との通信レイテンシが小さくなる利点がある。またその情報を使うことで、ALU-Net の Node 間でレジスタ値を Forwarding できるようになる。転送経路には Forwarding Wire を図 5 のように設ける。ここでは隣接 IB 間のレジスタ依存を対象とする 2 方式 (broadcast 方式と producer driven 方式) 提案する。

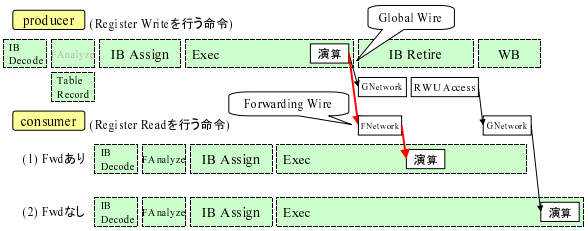

図 7: broadcast 方式のタイミングチャート

5.2 broadcast 方式

この方式は IB 間レジスタ依存の解析結果を consumer の Node に与え、受信側で入力値を Select する方式である。consumer だけがレジスタ依存に関する情報を持っているので、本方式は従来型の Fowarding と類似した機構であると言える。

5.2.1 Forwarding 情報の生成

本稿ではレジスタ値を出力する側の情報を producer 情報、入力する側の情報を consumer 情報と呼ぶ。それぞれ図 6 のように、レジスタ出力のあるレジスタ番号に対応するフィールドに 1bit の w-flag を持つマップを producer 側に、レジスタ入力のあるレジスタ番号に対応するフィールドに 1bit の r-flag を持つマップを consumer 側に、それぞれ静的に付加しておく。r-flag を持つマップには、レジスタ読み込みを行う命令の配置情報を座標として一緒に付加しておく。各論理レジスタ番号に対応したエントリのビットフラグの積が 1 であると、その 2 つの命令間に隣接 IB 間レジスタ依存が存在する。図の例ではレジスタ R2 を Forwarding の対象とみなし、Forwarding 情報として consumer の Node(1,1) に Assign する。

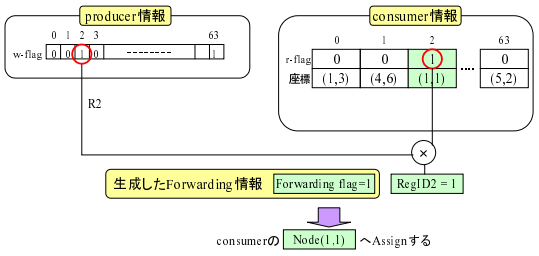

図 6: Forwarding 情報の生成

5.2.2 タイミングチャートの解説

タイミングの様子を図 7 に示す。各ブロックは処理ステージのタイミングを表し、ブロックの長さとパイプライン段数とは無関係である。Forwarding ありの場合 (1) となしの場合 (2) を併記する。

producer は IB Decode 終了後にレジスタ出力情報を Fwd Table に記録する (Table Record)。consumer は新たに FAnalyze(Forwarding Analyze) ステージへ進み、このステージで producer と consumer のレジスタ入出力情報をマージして Forwarding 情報を生成する。ここで得られた Forwarding 情報とそれ以外の Decode 情報を一緒に consumer の命令に付加し、次のステージ (IB Assign) へと実行を進める。

(2) の場合、producer の演算が終了すると、通常のレジスタ書き込み処理が始まる。通常処理ではレジスタ値を Global Network を経由して RWU へと送信する (GNetwork)。そして RWU 内でレジスタ依存関係を解析する (RWU Access)。解析の結果、後続 IB と IB 間データ依存の関係にあると判断された場合は、再び Global Wire を経由して目的の Node へデータを送信する (GNetowork)。

(1)Fwd ありの broadcast 方式では、producer の演算が終了した時点でレジスタ出力値を Forwarding ネットワークに送信する。転送経路には Forwarding ネットワークを用いる (FNetwork)。転送されたレジスタ値を consumer 側で受信すると、Forwarding 情報を使って自分に送られてきた値かどうかを Select する。ネットワークは RWU を経由しない Fowarding Wire を使う (FNetwork)。

また本方式では、producer からレジスタ値が broadcast された時点で、consumer の Assign がまだ完了していない場合も存在する。Assign 完了前の consumer には何を入力とすべきかの情報が存在しないため、送られてきたレジスタ値を Select することができない。そのため、受信したレジスタ値をそのまま破棄するか、もしくは Assign 完了まで Node 内に保持しておくかの 2 方式がある。レジスタ値を破棄する場合は単純で、レジスタ値を受信した時点で Assign が間に合った consumer のみ、レジスタ値を Select して Forwarding を行う。間に合わなかったものは通常の RWU 経由でレジスタ値を受信する。一方 Node 内に保持しておく場合には Node に Buffer を設け、宛先不明のレジスタ値はこの Buffer 内で Assign 完了まで待たせるようにする。いずれ

の方式が良いかを判断するため、両方式で測定を行ってみる。

5.3 producer driven 方式

2 つめの提案方式では、IB 間レジスタ依存関係を解析し、その結果をレジスタ値の送信元となる IB 側の命令に与える。Forwarding すべきレジスタ値を送信元で選択することで、Forwarding ネットワークの負荷を軽減できる利点がある。

5.3.1 Forwarding 情報の生成

broadcast 方式と同様に、図 8 の w-flag のマップを producerIB に、r-flag のマップを consumerIB にそれぞれ静的に付加しておく。各論理レジスタに対応するビット列とその命令の配置情報を保持するマップを保持する。解析ステージでは各エントリの比較を行い、IB 間レジスタ依存を判断する。図の例では R2 が IB 間レジスタ依存にあるので、consumer の座標情報 (1,1) を producer に送信 (Assign) する。

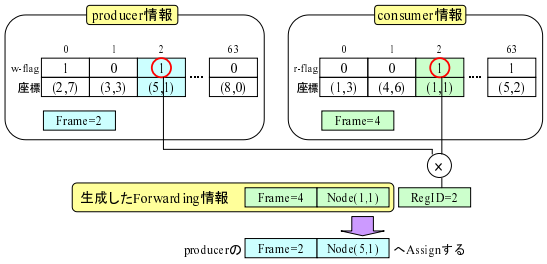

図 8: Forwarding 情報の生成

5.3.2 タイミングチャートの解説

本方式のタイミングチャートを図 9 に示す。broadcast 方式と異なる点は、FAnalyze ステージが IB Assign と同じタイミングで始まり、生成した Fowarding 情報を FAssign ステージで producer へ付加する点である。

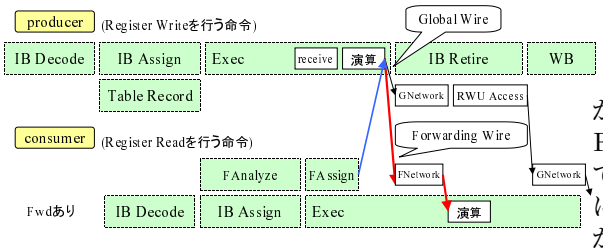

図 9: producer driven 方式のタイミングチャート

5.3.3 producer driven 方式の特徴

本方式の特徴は、後続 IB へ高速に転送すべきレジスタ値とそうでないレジスタ値を producer 側が知っていることにある。高速に転送すべきレジスタ値は IB 間の依存距離が 1 のレジスタ依存であり、この依存関係にあるレジスタ値のみを隣接 IB へ Forwarding Wire を使って高速に転送することで、Forwarding ネットワークへの負荷を抑えられるようになっている。この「高速に転送すべきレジスタ値を選択」することが本方式の利点であると言える。

6 性能評価と考察

本章ではそれぞれの方式をシミュレータに実装して性能の変化を調べ、考察する。

6.1 評価環境

VLDP3compiler でコンパイルした SPECint 95 に train 入力を使い、トレースベースの VLDP3simulator で評価を行った。IB は最大 64 命令を含むように作った。ここではモデルの基本的傾向を知ることを目的とするため、ALU-Net のトポロジは完全結合網を仮定した。完全結合網とは、ALU-Net 内の全ての Node が他の任意の Node と Local Wire で接続されたネットワーク・モデルである。VLDP3compiler の作成と ALU-Net トポロジによる IB の品質については関連研究で述べられている [7]。シミュレーションモデルは第 2.4 節の実行モデルを用い、分岐予測は 100 %ヒットとした。その他のパラメタは表 1 の通りである。前処理は Fetch、Decode、Assign でそれぞれ 2 サイクルと設定した。ROB はリオーダバッファサイズである。

表 1: 測定環境に用いたパラメタ

パラメタ	No-fwd	B	P
IB 多重度	8	8	8
ROB size	16	16	16
前処理 latency cycle	6	6	6
追加 FAnalyze latency cycle	0	1	1
追加 FAssign latency cycle	0	0	2
実行 latency(per Node) cycle	1	1	1
Local network latency cycle	0	0	0
Register access latency cycle	2	0	0
Register network latency cycle	2+2(往復)	0	0
Forwarding network latency cycle	0	1	1
Cache access latency cycle	1	1	1
Cache network latency cycle	0	0	0
Cache size	∞	∞	∞

各方式の性能比較を行うため、方式ごとに値が異なる要素のあるパラメタについて注目する。Forwarding ネットワークは現時点で理想的なので、方式ごとに変わることはなく、両方式とも 1 に設定する。そのかわりレジスタ・レイテンシが 0 になる。レジスタ依存の処理もマップのエントリ比較なので、処理量は同じである。よって FAnalyze レイテンシも同じに設定する (=レ

イテンシ 1)。FAssign レイテンシは ALU-Net への命令 Assign と同じ処理なので 2 に設定する。

6.2 各方式の性能比較

方式別に性能を測定した結果を図 10 に示す。No-forward は Forwarding を行わなかった場合、Broadcast-nobuffer と Broadcast-idealbuffer はそれぞれ Buffer なしと理想的 Buffer ありの broadcast 方式、Producer は producer driven 方式、Perfect-forward は理想的な Forwarding を行ったときの IPC である。また producer driven 方式で Forwarding 対象命令数を測定した結果を図 11 に示す。all depreg は、consumer から見たときに IB 間レジスタ依存がある命令数の IB あたりの平均値を表しており、fwd depreg は all depreg の中で Forwarding した命令数の平均を表している。D1 は隣接 IB の Forwarding だけをサポートした場合、D16 は ROB と同じだけの依存距離を Forwarding した場合の値である。

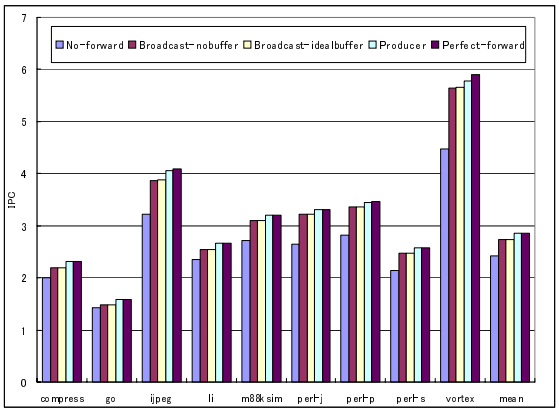

図 10: 各方式の IPC

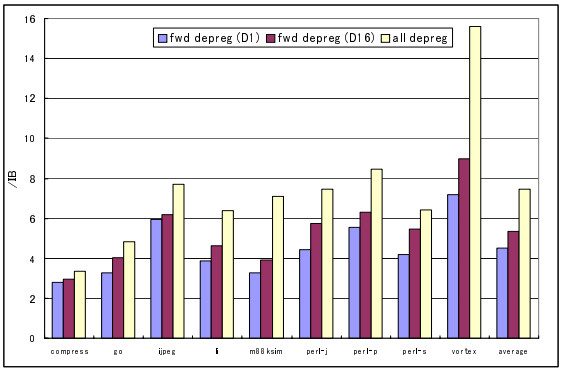

図 11: IB あたりに Forwarding した命令数

6.3 考察

図 10 によると、ベースラインとなる No-forwad の IPC と比較して、理想的な Forwarding を行った場合 (Perfect-forward) には約 2 割の性能向上が見込める。これを性能の上限値とすると、提案した producer driven 方式において理想的な値に近い性能が得られることがわかる。broadcast 方式との比較で見てみると、いずれも producer driven 方式の性能が上回っている。これは broadcast 方式で IB の実行ステージ中に FAnalyze ステージが追加され、パイプライン段数増えたことによると考えられる。

次に図 11 によると、IB あたり D16 の場合で約 3 割、隣接 IB 間にすれば約 4 割の Forwarding の対象命令数が削減できた。producer driven 方式は producer がレジスタ値を Forwarding Wire に流すべきかそうでないかを知っているので、この性質を使うことで Forwarding 対象命令数が削減でき、ネットワーク負荷を抑えられることが言える。

以上より、producer driven 方式で理想値に近い値が得られ、かつ、Forwarding 対象命令数を抑えてネットワーク負荷を軽くできるという点から考えて、提案方式の有効性が示されたと言えよう。

7 おわりに

本稿では多数 ALU を規則的に配置した実行機構を持つ VLDP3 アーキテクチャについて述べた。IB 間レジスタ依存の関係にある命令間でレジスタを介さない Forwarding 機構を提案し、その評価を行った。特に隣接 IB 間のレジスタ依存にのみ着目し、提案手法を用いることで、Forwarding 対象命令数を約 4 割削減することができた。性能向上率も理想値に近い値が得られることがわかった。今後は、より具体的な Forwarding ネットワークのトポロジを考える必要がある。

謝辞

本論文の研究は、一部、21 世紀 COE 情報技術戦略コア研究費によります。

参考文献

[1] 田中裕治, “VLDP3 アーキテクチャの構想 (3)〜レジスタフォワーディング機構の初期検討〜”, 情報科学技術フォーラム 2002 講演論文集 C-17, Sep. 2002.

[2] 入江英嗣, “VLDP3: データフローを高速実行する大規模アーキテクチャ”, 情報処理学会研究会報告 ARC 2003-ARC-151, Vol.2003, No.10, pp.49-54, Jan. 2003.

[3] 服部直也, “VLDP3 アーキテクチャに対するコード生成の検討”, 情報処理学会研究会報告 ARC 2003-ARC-151, Vol.2003, No.10, pp.55-60, Jan. 2003.

[4] 山口健輔, “VLDP3 アーキテクチャにおけるメモリリネーミング手法の検討”, HOKKE2003, 2003.