計算機アーキテクチャ 152−22
（２００３．３．１１）

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

山口 健輔 † 入江 英嗣 † 服部 直也 † 坂井 修一 † 田中 英彦 †

概要

近年、次世代アーキテクチャとして VLDP3 のように多数の ALU を規則的に並べたアーキテクチャが提案されている。アウトオブオーダ実行を行うアーキテクチャではメモリアクセス時に依存解決、アドレス計算など様々なオーバヘッドが生じるため、メモリアクセスの高速化は VLDP3 の性能を大きく左右すると考えられる。本稿では、メモリアクセス高速化手法の一つであるメモリリネーミングを VLDP3 に適用する機構として Speculative Wire Promotion を提案し、既存の手法である Speculative Memory Cloaking と併せてシミュレータに実装し、評価を行った。

Memory Renaming Mechanisms on VLDP3 Architecture

Kensuke Yamaguchi§ **Hidetsugu Irie**§ **Naoya Hattori**§
Shuichi Sakai§ **Hidehiko Tanaka**§

Abstract

In recent years, the architecture which put many ALUs in order like VLDP3 has been proposed as the next-generation microprocessor architecture. In the architectures which adopt out-of-order execution, performance improvement is disturbed by large latency generated by dependence analysis, address calculation, and so on, so acceleration of memory access is likely to influence performance of VLDP3 greatly. In this paper, we describe the instruction execute model and memory access mechanism of VLDP3 Architecture, and propose ”Speculative Wire Promotion” as a mechanism in which we mount Memory Renaming Mechanisms in VLDP3 Architecture. We also implement it on VLDP3 simulator with the known mechanism ”Speculative Memory Cloaking”, and evaluated them by simulation results.

1 はじめに

半導体デバイス技術の進歩によってマイクロプロセッサが利用可能な半導体資源は年々増加を続けており、数年後にはさらに豊富なトランジスタ資源を利用できると予想されている。しかし、スーパースカラに代表される従来のアーキテクチャをそのまま大規模化しても並列性の抽出に限界があり、十分な性能向上ができない。そこで、大規模並列実行を可能とする次世代アーキテクチャとして VLDP3(Very Large Data Path) アーキテクチャの研究開発が行われている。

アウトオブオーダ実行を行うプロセッサは、ロード命令に先行するストア命令の中に参照アドレスが確定していないストア命令が存在する場合、ロード命令のキャッシュアクセスを遅らせる必要があり、この待ち時間がオーバヘッドとなってしまう。その他にも、メモリアクセス時にはアドレス計算やメモリユニットとの通信レイテンシなど様々なレイテンシによるオーバヘッドが生じるため、メモリアクセスの高速化は VLDP3 の性能を大きく左右すると考えられる。そこで本稿では VLDP3 の命令実行モデル及びデータ供給の概略を述べ、メモリアクセス高速化手法の一つであるメモリリネーミングを VLDP3 に適用

† 東京大学
§ University of Tokyo

する機構として Speculative Wire Promotion を提案し、既存の手法である Speculative Memory Cloaking と併せてシミュレータに実装し、評価を行う。

2 関連研究

本章ではメモリアクセスによるレイテンシを隠蔽する手法の一つであるメモリリネーミング [8] を実現するための機構として、Speculative Memory Cloaking[4, 5] を紹介する。

Speculative Memory Cloaking では、DDT (Dependence Detection Table)、DPNT (Dependence Prediction and Naming Table)、SF(Synonym File) と呼ばれる 3 種類のテーブルを用いる。図 1 に、これらのテーブルを用いてメモリ依存の予測及び値のフォワーディングを行う様子を示す。

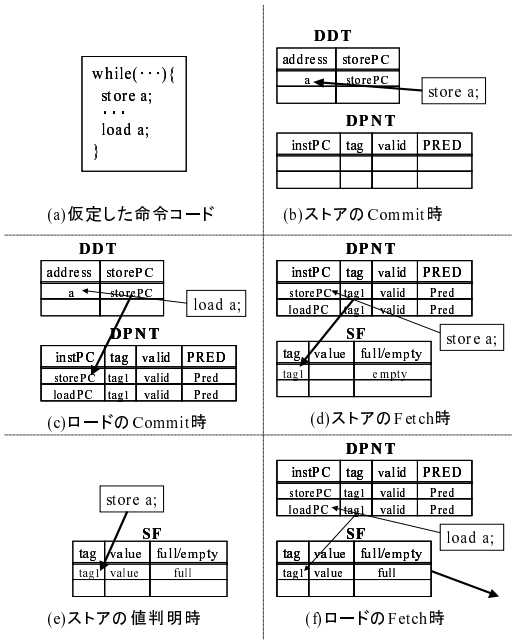

図 1: Speculative Memory Cloaking の動作

図 1-(a) に示すような、メモリ依存関係にあるストア命令とロード命令が繰り返し実行されるようなコードを仮定する。

まず、ストア命令は Commit 後に DDT に PC とデータのアドレスを書き込む (図 1-(b))。

ロード命令は Commit 後に DDT の Address フィールドを参照し、同じアドレスにアクセスしたストア命令のエントリが存在したら DPNT にストア、ロード両方の命令のエントリを作り、それぞれの PC を記録する。また、共通の tag を生成して両方のエントリに書き込み、valid bit を valid に設定する (図 1-(c))。

ストア命令はフェッチされたときに毎回 DPNT の PC フィールドを参照し、依存予測情報を発見したら SF に tag を書き込んで full/empty bit を empty に設定する (図 1-(d))。そしてストア値が判明したらストア値を SF に書き込むと同時に full/empty bit を full に設定し (図 1-(e))、通常のキャッシュアクセスを行う。

ロード命令もフェッチされたときに同様に毎回 DPNT を参照し、依存予測情報を発見したら DPNT に記録されている tag をインデックスとして SF を参照し、full/empty bit が full となっていたら記録されているストア値を引き出し、ロード命令の Consumer に送る (図 1-(f))。その後ロード命令は非投機的なキャッシュアクセスを行い、正しいロード値と予測値を比較する。一致していなかったら、ロード命令以降の処理をやり直す。

3 VLDP3 の概要

本章では、VLDP3 の命令実行方式およびキャッシュアクセス方式について説明する。

3.1 VLDP3 の命令実行方式

実行コードは IB(Instruction Block) という単位で与えられ [2]、この IB を ALU-Net と呼ばれる命令実行機構に割り付けて命令を実行する。以下で IB 及び ALU-Net について述べる。

3.1.1 ALU-Net

VLDP3 の命令実行機構は、複数の ALU が網目状に並べられ、ごく短い配線 (Local Wire) で結ばれた ALU-Net 構造である。各ノードは Assign された命令に基づいて演算を行い、結果を Local Wire を用いて次のノードに受け渡す (図 2 右)。各ノードには同時に複数の命令を Assign することができる。Assign された命令は、Assign された順とは無関係にオペランドが揃ったものから発火する。ALU-Net 内と外部とのデータの受け渡しは Global Wire と呼ばれる配線を用いて行われ、Local Wire によるデータ通信に比べて、レイテンシが大きい。

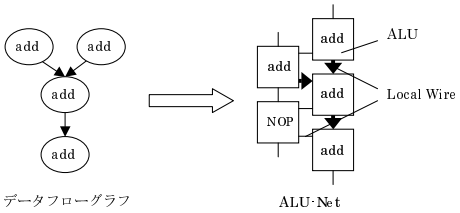

図 2: ALU-Net への命令 Assign の様子

3.1.2 IB

大規模演算機構 ALU-Net 制御の単純化のために、VLDP3 では命令割り当てや解放を、複数の命令をまとめた塊を最小単位として行う。この塊を IB と呼ぶ。同一 IB 内の命令に関しては、依存するデータを全て Local Wire で転送することを想定しており、これを保証するために IB はコンパイラが生成する。コンパイラはプログラム中のデータフローを解析し、ALU-Net で実行できるように各命令の割り当て位置を決定する。

3.2 VLDP3 のキャッシュアクセス方式

VLDP3 では、ロード・ストア間の依存関係の解析及びキャッシュのアクセス順序の制御を LSU(Load Store Unit) と呼ばれるユニットで行う。IB の Fetch 要求が送信されてからロードの Consumer にロード値が届くまでの、依存関係にあるストア命令とロード命令のパイプライン処理の様子を図 3 に示す。

図 3: 依存関係にあるストア命令とロード命令のパイプライン処理の様子

まず分岐予測器から Fetch Unit に Fetch 要求が送信され、IB が Fetch される。次に Decoder に渡されて Decode され、ALU-Net に Assign される。ストア命令はストア値、アドレスのオペランドが届いてアドレス計算が終わると LSU にキャッシュ書き込み要求を送信する。ロード命令はアドレスのオペランドが届いてアドレス計算が終わると LSU にキャッシュ読み出し要求を送信する。LSU では、ストア命令及びロード命令のキャッシュアクセスは命令コード中で自らに先行するすべてのストア命令及びロード命令がキャッシュにアクセスした後で行うように制御されるので、ロード命令は依存関係にあるストア命令がキャッシュへの書き込みを終えた後でキャッシュから値を読み出して ALU-Net にあるロード命令の Consumer のノードに値を送る。

なお、VLDP3 では Wait Mask Table[3] と呼ばれるメモリ依存予測手法を用いて投機的なロード処理も行っているが、簡単のため以下では非投機ロードを基準に議論する。

3.3 VLDP3 の命令実行モデル

本節では、VLDP3 の命令実行モデルについて述べる。IB が Fetch されてから Retire するまでのパイプライン処理を図 4 に示す。

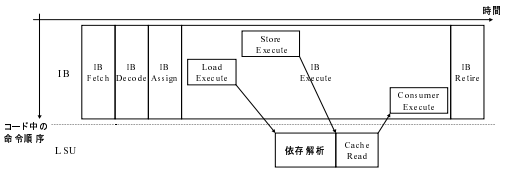

図 4: IB を単位とした命令実行のパイプライン処理

まず命令は IB 単位で Fetch、Decode され、得られた配置情報を元に ALU-Net に Assign される。Assign が完了した IB の各命令は、IB Execute ステージでは全て命令単位で実行される。各命令はオペランドが揃った時点で発火し、演算を開始する。ロード命令、ストア命令はオペランドが揃ってアドレス計算が終わった時点で LSU にキャッシュアクセス要求を送信し、依存解析が終わってからキャッシュにアクセスする。ロード命令のキャッシュ読み出しが終わるとロード値はロード命令の Consumer のノードに転送され、Consumer のノードでの演算が開始される。IB 内の命令の演算が全て終わると ALU-Net 上から IB が flush され、新たな IB の Assign が可能となる。この状態において IB 内の全ての命令が例外を起こさないこと、及び直前の IB の Retire が確認されると、IB は Retire を開始する。

4 メモリリネーミングの VLDP3 への実装

本章では、VLDP3 におけるストア・ロード間のデータ転送を高速化することを目的として、メモリリネーミングの VLDP3 への実装を検討する。ストア・ロード間のデータ転送に Local Wire を用いることができれば極めて高速なデータの転送が可能であるが、Local Wire を用いたデータ転送はコンパイル時に指定する必要があるため、同一 IB 内のデータ供給にしか適用できない。そこで、依存するストア・ロードが同一 IB に含まれる場合 (IB 内メモリ依存) と異なる IB に含まれる場合 (IB 間メモリ依存) を区別して考え、IB 内メモリ依存を高速化する手法として Speculative Wire Promotion を提案する。尚 IB 間メモリ依存に対しては、関連研究として紹介した Speculative Memory Cloaking を適用して高速化する。

4.1 Speculative Wire Promotion の概要

Speculative Wire Promotion では IB 内メモリ依存を静的に予測し、コンパイル時にストア命令のノードからロード命令の Consumer のノードまで Local Wire を用いた投機的なフォワーディングパスを構築することによってロード命令のレイテンシを隠蔽する。Speculative Wire Promotion に対応した IB 生成の様子を図 5 に示す。

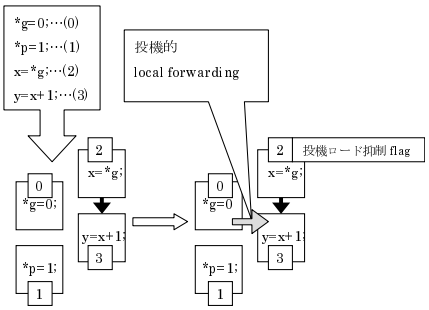

図 5: Speculative Wire Promotion に対応した IB 生成

以下で Speculative Wire Promotion の動作を示す。まず、コンパイル時に同じアドレスにアクセスするストア命令 (0 番) とロード命令 (2 番)、及びそのロード値を使用する命令 (3 番) を予測する。次に、検出されたストア命令からロード命令の Consumer まで Local Wire を用いた転送路を指定する。そして、0 番のストア命令のストア値が判明し次第 3 番のノードに値がフォワーディングされ、3 番の命令が演算可能となる。このことにより、0 番、1 番のストア命令と 2 番のロード命令によるレイテンシの影響を全く受けずに 3 番以降の命令を実行することができる。なお、0 番のノードからフォワーディングされた値は、後で正しい値と比較するために LSU に転送しておく。そして、0 番と 1 番の命令が実行された後に 2 番のロード命令が非投機的に実行されることにより正しい値が判明し、その値が LSU で投機的にフォワーディングされた値と比較される。一致すればそのまま後続の処理を続行し、一致しなければ、投機的フォワーディングパスを削除した上で IB の先頭から非投機的に処理をやり直す。投機が失敗していた場合は予測テーブルにそのストア命令の識別子を書き込み、以降はそのストア命令のノードからの投機的フォワーディングパスは動的に削除する。

4.2 Speculative Wire Promotion による命令処理のレイテンシの削減効果

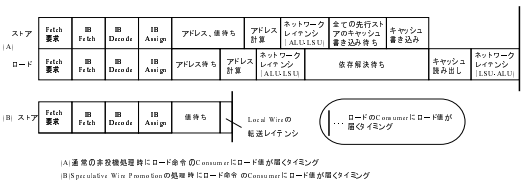

図 6: ロード命令の Consumer のノードにロード値が届くタイミング

Speculative Wire Promotion による処理を行った時にロード命令の Consumer のノードにロード値が早いタイミングで届くようになる様子をタイミングチャートを用いて図 6 に示す。また、非投機的にキャッシュアクセスを行った場合と Speculative Wire Promotion による予測ロード値のフォワーディングを行った場合のデータ通信の様子を図 7 に示す。

これらの図に示したように、非投機ロードによる値の転送では Global Wire を用いた値の転送を 2 回行うのに対し、Speculative Wire Promotion では Local Wire を用いた非常にレイテンシの小

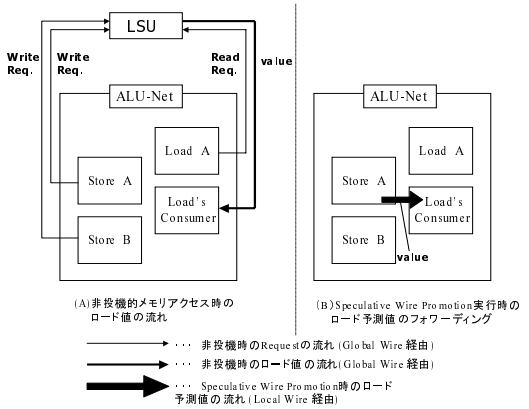

図 7: 非投機的キャッシュアクセス時と Speculative Wire Promotion 実行時のデータ通信の様子

さな転送を 1 回行うだけである。またストアとの依存解決を待つ必要がなく、ストア値が判明した時点で値をロード命令の Consumer のノードにフォワーディングする。このため、非常に早いタイミングでロード命令の Consumer が演算可能になる。

4.3 Speculative Wire Promotion の評価

本節では、VLDP simulator3 に Speculative Wire Promotion 及び Speculative Memory Cloaking を実装したときの性能の変化を示し、評価を行う。

4.3.1 評価環境

VLDP3compiler でコンパイルした SPECint95 のバイナリ 6 種類に 7 種類の train の入力を使い、トレースベースの VLDP3 simulator で評価を行った。VLDP3 compiler は、alpha をベースにした RISC 型命令列を最大 64 個まとめて 1IB とした。IB には ALU-Net への配置情報も静的に付加している。

シミュレーションモデルは第 3 章で述べた VLDP3 の命令実行モデルと同様である。ただし、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論文で述べられている [2]。

シミュレーション時の各種パラメタは表 1 の通りとする。

表 1: VLDP3 シミュレータの各パラメタ

パラメタ	値
IB 多重度	8
Reorder Buffer size	16
Fetch latency	2 cycle
Decode latency	2 cycle
Assign latency	2 cycle
Execute latency(per Node)	2 cycle
Local Wire network latency	0 cycle
ALU-Net → LSU network latency	3 cycle
LSU → ALU-Net network latency	2 cycle
ALU-Net → Cloaking Prediction Table network latency	3 cycle
Cloaking Prediction Table → ALU-Net network latency	2 cycle
Cloaking Prediction Table access latency	4 cycle
Cache access latency	4 cycle
Cache size	∞
Prediction Table size	∞
Branch Prediction	100% hit
Cache	100% hit

ROB size とはプロセッサの中に保持できる最大 IB 数である。ここでは 16 とした。Local Wire による転送レイテンシは非常に小さいため、0 cycle とした。ALU-Net と LSU、ALU-Net と Speculative Memory Cloaking の予測テーブルの間のネットワークレイテンシは等しいものとした。また、キャッシュと Speculative Memory Cloaking の予測テーブルの書き込み、読み出しのレイテンシも等しいものとした。なお、Speculative Memory Cloaking の予測テーブルのエントリ数及びキャッシュのサイズは無限大とし、キャッシュは 100%ヒットするものとした。また、分岐予測は 100%当たるものとした。

4.3.2 評価

次に図 8 にメモリリネーミングを行わない場合、Speculative Wire Promotion を実装した場合、理想的な Speculative Wire Promotion を実装した場合、Speculative Memory Cloaking を実装した場合、理想的な Speculative Memory Cloaking を実装した場合、Speculative Wire Promotion と Speculative Memory Cloaking を両方実装した場合の各アプリケーションにおける IPC、及びその調和平均を示す。ここで言う理想的な Speculative Wire Promotion とは、すべての IB 内メモリ依存についてストア値が判明し次第正しい値をロード命令の Consumer に Local Wire を用いて転送する機構であり、理想的

な Speculative Memory Cloaking とは、Speculative Memory Cloaking の機構においてロード・ストア間のメモリ依存をすべて正確に予測してロードの Consumer への値の転送を行うものである。

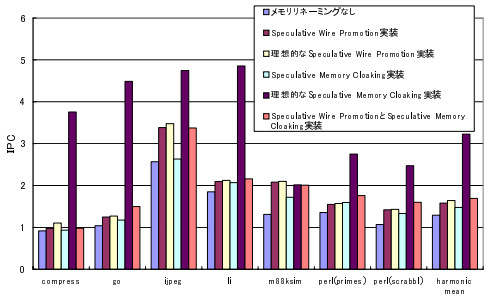

図 8: メモリリネーミングの実装と IPC の変化

Speculative Wire Promotion を実装した場合には平均 22%程度の IPC の向上が見られる。理想的な Speculative Wire Promotion を実装した場合との差は 4%程度である。Speculative Wire Promotion ではストア値が判明し次第すぐに Local Wire を用いて値をロード命令の Consumer のノードに転送しており、図 6 に示すように、通常のキャッシュアクセスにおいてロード命令の Consumer にロード値を送る過程でのレイテンシの大部分を削減している。よって理想的な Speculative Wire Promotion に近い IPC が実現できているということは、ストア命令と IB 内メモリ依存関係にあるロード命令のキャッシュアクセスのレイテンシを大部分隠蔽することができたと言える。

Speculative Memory Cloaking を単体で実装した場合では 14%程度の IPC の向上が見られるが、Speculative Wire Promotion と同時に実装しても大きな IPC の向上は見られない。これは、Speculative Memory Cloaking による依存予測が IB 間については外れやすく、高速化できている部分が Speculative Wire Promotion と重複しているためと考えられる。また理想的な Speculative Memory Cloaking との差が大きいという点でも依存予測があまり的中していないことが分かる。そこで今後は、IB 間メモリ依存について依存予測の精度向上を検討する。

5 おわりに

本稿では VLDP3 へのメモリリネーミング適用方式として Speculative Wire Promotion を提案し、Speculative Memory Cloaking と共に VLDP3 simulator に実装して IPC の変化を計測し、評価した。

Speculative Wire Promotion は、IB 内メモリ依存に対してそのロード命令のキャッシュアクセスのレイテンシの大部分を隠蔽できることが示された。それに対して Speculative Memory Cloaking はある程度の性能向上は見られたものの、まだ性能向上の余地が残されており、今後予測精度の向上を検討する必要があることが分かった。

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] 入江英嗣, 服部直也, 坂井修一, 田中英彦. ”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)

[2] 服部直也, 入江英嗣, 坂井修一, 田中英彦. ”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)

[3] 谷地田瞬, 山口健輔, 田中裕治, 服部直也, 入江英嗣, 飯塚大介, 坂井修一, 田中英彦. ”VLDP3 アーキテクチャの構想 (4)〜メモリ依存に関する初期検討〜”. 情報科学技術フォーラム 2002 講演論文集 C-17. (Sep. 2002)

[4] Andreas Moshovos and Guri Sohi.”Streamlining Inter-operation Memory Communication via Data Dependence Prediction”.In the Proc. of MICRO-30. (Dec. 1997)

[5] Andreas Moshovos and Guri Sohi.”Speculative Memory Cloaking and Bypassing”.Invited. To appear in the International Journal of Parallel Programming.(Oct. 1999)

[6] R.E.Kessler, E.J.McLellan, and D.A.Webb.”The Alpha 21264 Microprosessor Architecture”.in International Conference on Computer Design.(Dec. 1998)

[7] Mikko H.Lipasti, Christopher B.Wilkerson, and John Paul Shen.”Value Locality and Load Value Prediction”.in 17th International Conference on Architectural Support for Programming Languages and operating Systems, pp. 138-147.(Oct. 1996)

[8] Brad Calder, Glenn Reinman.”A Comparative Survey of Load Speculation Architectures”.Journal of Instruction-Level Parallelism 1.(2000)