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～

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[‡]

1. はじめに

コンピュータの処理の中核となるマイクロプロセッサのデバイス技術は、今後10年以上堅調な見通しであり、数億ものゲートから成る大規模プロセッサも現実のものとなりつつある。一方アーキテクチャ技術は、コンピューティングの本質であるシングルスレッド処理に関しては、近年大きな変化は見られない。ここで我々は十数年先のデバイス技術を仮定した新しいアーキテクチャ、VLDP(大規模データパス)アーキテクチャを提案してきた [1][2][3]。

現在は VLDP アーキテクチャの最新モデルである VLDP3 アーキテクチャ[6] を提案しているが、メモリ通信に関して Load/Store 命令の処理能力が深刻なオーバヘッドとなることが予測される。一般に Load/Store 命令は全実行コード中の3割程度を占め、他の演算命令と異なり命令間の依存関係の有無がはっきりしないため、多数の命令を同時に扱うには様々な困難が伴う。そのため、大規模並列実行を行う VLDP3 アーキテクチャではメモリシステムが性能に極めて大きく作用すると予想される。

本稿では、VLDP3 アーキテクチャにおけるメモリアクセス処理の高速化を目的とし、特に Load/Store 依存に着目し、VLDP3 アーキテクチャに適した Load/Store 依存予測機構を提案する。また、メモリ依存が予見可能な場合の Forwarding の高速化に関する予備評価を行うものとする。

2. メモリ依存予測手法の紹介

2.1 Blind Load Prediction

Blind Load Prediction は最も単純なメモリ依存予測手法であり、全ての Load 命令は Store 命令に依存がないと仮定して投機的に実行する。つまり、Load のアドレス計算が終わってアドレスが判明した時点で投機的に Load 命令が実行され、メモリへのアクセスを開始する。

しかし、先行する Store 命令と依存関係が存在した場合は、Load 命令を先に発行してしまうと、逐次実行した場合と結果が一致しなくなることがある。そこで、依存関係の違反を検出した場合、パイプラインフラッシュによるプログラムステートの復元の後、問題の Load 命令以降を再実行する必要がある。投機実行のプロセッサにおいてパイプラインフラッシュは大きなペナルティを生じるため、極力避けなくてはならない。

2.2 Wait Table Prediction

Wait Table Prediction では、予測ミスが発生するまで Blind Load Prediction 同様に Load 命令を発行し続ける。しかしここで Blind Load Prediction と異なる点は、再び予測ミスが発生することを回避するために、Load 命令の PC をインデックスとして Wait Table の State Field にフラグを立て、依存の有無 (0:依存なし,1:依存あり) を表現する。初期値は値 0 とする。

次回の実行からは、Load 命令をフェッチする際に Wait Table を用いて、以前に予測ミスを引き起こした Store 命令が実行中であるかを検出する。過去に予測ミスを起こした Load 命令の Store 命令の追い越しを禁止することで、予測ミスを回避する。尚、State Field は一定間隔で 0 clear する。Wait Table の構成を図1に示す。この機構は Alpha21264Processor にも実装されている [4][5]。

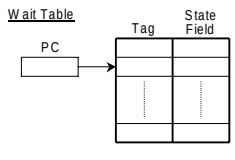

図 1: Wait Table

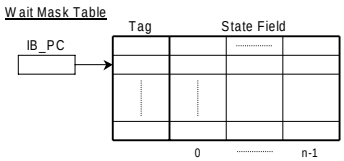

図 2: Wait Mask Table

3. VLDP3アーキテクチャにおけるメモリ依存予測機構

VLDP3 アーキテクチャにおいて、メモリ依存の有無を予測するために Wait Table を改良した Wait Mask Table を用いる。Wait Mask Table の構成を図2に示す。(図2では IB 内の命令数は n 命令と仮定する)

VLDP3 アーキテクチャは、複数命令を Instruction Block(以下 IB と呼ぶ) 単位で管理 [6][7] しているため、Wait Table を採用した場合、次のような問題が生じてしまう。

- 命令単位で Wait Table を読み書きしようとするとポート数が多くなってしまう
- Wait Table を IB_PC(IB に静的につけられたプログラムカウンタ) だけで indexing しようとすると、命令単位ではなく IB 単位でしか制御できず、IB 内に含まれる全ての Load 命令に同一の指令 (wait または not) しか送れない

Wait Mask Table では IB_PC で indexing する。State Field には n ビットの情報が記憶でき、それが IB 内の n 命令に対応する。動作は Wait Table と同様で、State Field のビットに値 1 が立っていれば、データ依存関係があることを予測し投機実行を止める。

VLDP3 アーキテクチャでは命令を IB 単位で管理していることから、Wait Mask Table は各 Load 命令毎に管理するよりポート数が少なく済む。また IB 内の各命令それぞれに指令を送ることが可能となる。これで上記に挙げた 2 つの問題を解決できる。

[†]（株）日立製作所 日立工業専門学院
[‡]東京大学大学院 情報理工学系研究科
[*]東京大学大学院 工学系研究科
[§]東京大学工学部 電子情報工学科

4. VLDP3アーキテクチャにおけるメモリ通信の傾向分析

4.1 メモリ依存の IB 間距離の頻度

第 3 節で述べた Wait Mask Table を用いた場合、多くのメモリ依存の有無は予測可能であると考えられる。次に、メモリ依存があると予測した場合どのように高速な Forwarding を実現するかが課題となる。

そこで依存のある IB 間の距離に着目してその頻度分布を調べ、局所性から最適化すべき距離を特定できるか検討してみる。

評価環境として、最適化コンパイラ mewcc を使いベンチマークとして SPECint95 の 7 つのアプリケーションを用いた。また、Profile を用いて作成した各 binary に test を入力して測定した。本評価では、2001 年度までの VLDP(VLDP2) アーキテクチャで使用していた IB を用いた。この IB は最大 4Basic Block で構成されており、1Basic Block は最大 8 命令を含むことができる [3]。

図 3 にメモリ依存の IB 間距離毎の頻度を示す。尚、図 3 は SPECint95 の 7 つのアプリケーションの結果の平均を取ったものである。

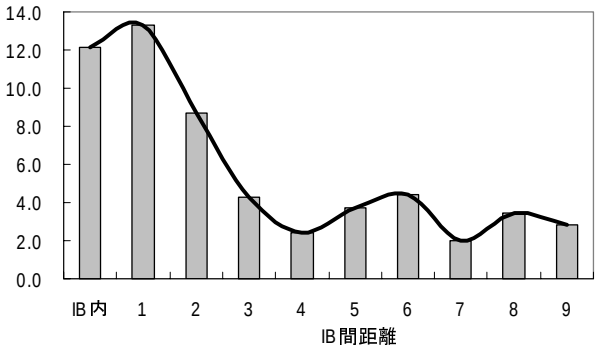

図 3: メモリ依存の IB 間距離毎の頻度 [%]

図 3 より、IB 内メモリ依存及び隣接 IB 間メモリ依存の占める割合が多いことが分かる。

よって、依存頻度の高い IB 内メモリ通信及び隣接 IB 間メモリ通信に注目し、高速 Forwarding の機構を検討していくことにする。尚、隣接 IB 間メモリ依存については今後の課題とする。

4.2 IB 内メモリ依存の静的解析による高速 Forwarding と予測ヒット率の向上

メモリ通信のハードウェア設計には、一方を追求すれば他方を犠牲にせざるを得ないという二律背反の関係があり、動的な手法だけでは全体的な性能向上を目指すことが難しい。ここで、動的手法の手助けとして、更なる性能向上にはコンパイラを利用した静的なアプローチが不可欠だと考える。

コンパイラは、Load/Store 命令がメモリの同アドレスにアクセスするという情報は、ある程度コード中から静的に判別することができる。同アドレスにアクセスする Load/Store 命令を検出し、Load 命令の実行を待たずに Store 命令の時点で、値を後続の命令に渡すことで高速に Forwarding できると考えた。つまり Store 命令とその値を使う後続の命令との間に、Local Wire を用いた投機的な Forwarding パスをコンパイル時に構築することによってメモリ通信を高速化できる。

また、コンパイル時、検出された Load 命令に非投機フラグを立て投機実行しないように設定する。この手法によって第 3 節で述べた Wait Mask Table の初期値を静的に指示でき、初回のミスを防ぐことが可能となる。更に、静的に指示した部分については、State Field を 0 clear した直後も残しておくことができる。

この手法の初期評価のために、コンパイラで静的にメモリ依存解析できそうな依存のうち、Constant Address を介した依存と関数内の Stack Address を介した依存で IB 内メモリ依存の割合を調べた。その結果を図 4 に示す。

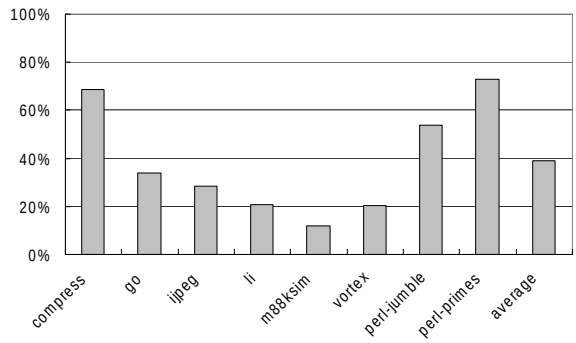

図 4: 静的に予見可能な IB 内メモリ依存の割合

図 4 より IB 内メモリ依存の約 40%は静的に解析できることが分かった。今後の課題として、コンパイラでの解析能力を向上させ、高速 Forwarding 機構の検討をしていく。

5. おわりに

本稿では、VLDP3 アーキテクチャに適した Wait Mask Table Prediction におけるメモリ依存の有無を予測する機構を提案した。更にメモリ通信の傾向から、IB 内及び隣接 IB 間を注目すべき距離とし、今回は IB 内依存の静的解析による高速 Forwarding の可能性を示した。

今後の課題として、IB 内の高速 Forwarding のための更なる静的支援と合わせて、IB 間 (特に隣接 IB 間) の高速 Forwarding 機構を検討していく。

参考文献

[1] 田中英彦. ”ここいらで、計算機アーキテクチャを再考しよう”. 情報処理学会研究報告 計算機アーキテクチャ研究会. 94-ARC-108, No.91, pp.33-40(1994)

[2] 中村友洋, 吉瀬謙二, 辻秀典, 安島雄一郎, 田中英彦. ”大規模データパス・プロセッサの構想”. 情報処理学会研究報告 計算機アーキテクチャ研究会, 97-ARC-124, No.61, pp.13-18(1997)

[3] 辻秀典, 安島雄一郎, 坂井修一, 田中英彦. ”大規模データパス・プロセッサの提案”. 情報処理学会研究報告 計算機アーキテクチャ研究会. 2000-ARC-139, No.74. pp.55-60(2000)

[4] Alpha 21264 Microprocessor Hardware Reference Manual. COMPAQ COMPUTER CORPORATION

[5] G.Reinman and B.Calder. ”Predictive Techniques for Aggressive Load Speculation”. IEEE MICRO-31, pp.127-137(1998)

[6] 入江英嗣, 山口健輔, 谷地田瞬, 田中裕治, 服部直也, 飯塚大介, 坂井修一, 田中英彦. ”VLDP3 アーキテクチャの構想 (1)〜プロセッサ構成〜”. FIT2002. (2002)

[7] 服部直也, 山口健輔, 谷地田瞬, 田中裕治, 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 (2)〜ソフトウェア支援〜”. FIT2002. (2002)