1J-4

ALU カスケーディングと先行実行を併用するプロセッサ

松村 貴之 † 吉瀬 謙二 †
東京工業大学大学院情報理工学研究科 †

1 はじめに

現在のプロセッサはメモリウォール問題をオウトオブオーダー実行によって緩和している．しかし，その為には多くのハードウェアが必要である．

本稿では，よりハードウェアコストが小さく，性能向上を目指す手法として ALU カスケーディングと先行実行を併用するプロセッサを提案する．

2 ALU カスケーディング

ALU カスケーディングとは，ALU の出力を別の ALU の入力へフォワーディングすることで，依存関係にある二つの命令を 1 クロックサイクルで実行する手法である．図 1 の Inst2 が Inst1 の結果を用いる場合，ALU2 の直前のマルチプレクサによって ALU1 の結果を選択することで，Inst1 と Inst2 が 1 サイクルで実行することができる．

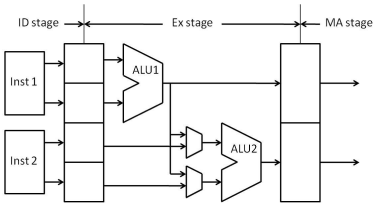

図 1: ALU カスケーディング

3 ALU カスケーディングの性能評価

ALU カスケーディングの評価環境を示す．シミュレータとして，SimMips[1] をサイクルアキュレートに拡張した SimMipsSS に ALU カスケーディングを実装したものを用いる．ALU カスケーディングを適用しないインオーダースーパースカラをベースモデルとする．ベンチマークアプリケーションとして SPEC2006 の整数ベンチマーク 6 種類を用いる．データセットには train を用い，1G 命令スキップ後の 100M 命令を評価する．プロセッサのパラメータを表 1 に示す．

図 2 の normal に IPC 向上率，図 3 に ALU カスケーディング適用率を示す．適用率とは,「他の ALU の結果を受け取って実行した命令の数を全実行命令数で割った値」と定義する．

基本的に ALU カスケーディング適用率が大きいほど IPC 向上率が高いことが分かる．しかし，mcf のように ALU カスケーディング適用率が高くても IPC 向上率が小さい場合がある．この原因として，キャッシュヒット率が低いことが考えられる．パイプラインがストールすることで総実行サイクルが多くなってしまうため，ALU カスケーディング適用率が大きく，命令実行のサイクル数を減少させても性能向上率は小さくなる．

表 1: 評価に用いたプロセッサのパラメータ

Pipeline	IF, ID, EX, MA, WB
L1 I-cache,	32KB 4-way 64B line
L1 D-cache,	32KB 4-way 64B line
L2 cache,	4MB 8-way 64B line 25 cycle hit latency
Memory latency	300 cycle latency

キャッシュヒット率を図 4 に示す．命令キャッシュのヒット率ははすべてのベンチマークでほぼ 100% であったが，データキャッシュの場合はベンチマークによってばらつきがある．mcf の場合は約 84% と最もヒット率が低いことが分かる．

データキャッシュミスのストールが 0 サイクルの場合の ALU カスケーディングによる IPC 向上率を図 2 の ideal Dcache に示す．カスケーディング適用率が大きかったもの (bzip2，hmmer) や，キャッシュヒット率の小さいもの (mcf) の向上率が特に大きくなっているのが分かる．

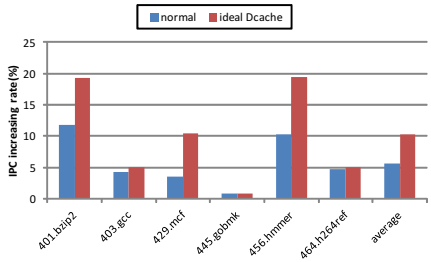

図 2: IPC 向上率

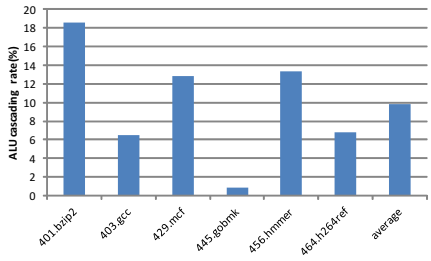

図 3: ALU カスケーディング適用率

A processor which combines ALU cascading with Instruction Pre-Execution
Takayuki MATSUMURA and Kenji KISE
† Graduate School of Information Science and Engineering, Tokyo Institute of Technology

Copyright ©2012 Information Processing Society of Japan.
All Rights Reserved.

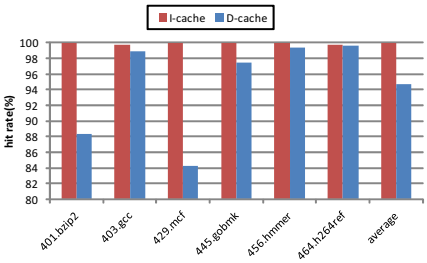

図 4: キャッシュヒット率

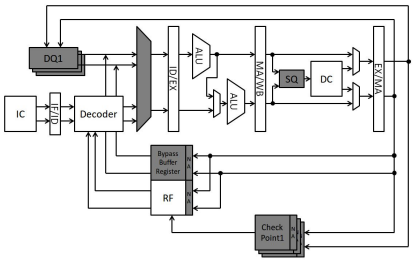

図 5: 提案するプロセッサのブロック図

```
Inst1   add  $3   $1   $2(NA)
Inst2   add  $5   $4   $3(NA)
Inst3   add  $7   $6   $5(NA)
Inst4   add  $9   $8   $7(NA)
```

図 6: ALU カスケーディングと組み合わせることで効率的に実行できる命令列

4 先行実行

データキャッシュミスによるストールサイクルを減少させることで ALU カスケーディングによる性能向上率が大きくなることが分かった．そこで，データキャッシュミス時に後続の命令を先行実行していく手法を組み合わせることを提案する．

4.1 Simultaneous Speculative Threading

ROCK プロセッサ [2] で用いられている先行実行手法 (Simultaneous Speculative Threading，SST) について説明する．SST では，キャッシュミスが起きてもストールせずに投機的に実行していく命令列 (Ahead thread) とキャッシュミスを起こした命令とそれに依存した命令列 (Behind thread) を同時に処理する．

キャッシュミスを起こした命令とそれに依存する命令は DQ(Deferred Queue) と呼ばれるキューに格納していき，キャッシュミスが解決された後でもう一度実行していく．依存しない命令はレジスタに値を書き込んでリタイアする．

Behind thread が完了すると，Behind thread から Ahead thread へ必要なレジスタの値を渡すためにスレッドの同期を行う．

命令やレジスタファイルのエントリは NA(Not Available) ビットを持ち，これによってデータハザードを回避している．

4.2 提案手法

提案するプロセッサのブロック図を図 5 に示す．先行実行のためのモジュールを灰色の部分で示している．SST は Ahead thread と Behind thread を同時に実行する SMT(Simultaneous Multi-Threading) であるが，この 2 つのスレッドを排他的に実行する VMT(Vertical Multi-Threading) によって先行実行を行う．

VMT にすることで SMT の場合に必要であったスレッドの同期が必要無くなり，そのために必要なサイクル分高速化できる．

SST と比較して，非投機な命令を実行する場合は ALU カスケーディングを用いるため，提案手法のほうが高性能である．Behind thread を実行する場合，SST では Ahead thread を同時に実行している．そのため依存関係にない命令を発行できるので演算器を効率的に利用できる．提案手法の場合では，依存関係があったとしても ALU カスケーディングによって依存関係を解決するため，演算器の利用効率は SMT の場合に比べてあまり低下しないと考えられる．

VMT で先行実行を行う場合，ALU カスケーディングを適用しなければ，Behind thread を効率的に実行できない可能性がある．DQ に格納される命令は「データキャッシュミスを起こした命令」と「すでに DQ に格納されている命令のうち少なくとも一つの命令に依存した命令」であるため，DQ 内の命令列には通常の命令列と比較して多くの依存関係が存在する．図 6 のように 1 つ前の命令に依存する命令が連続して DQ 内に存在する場合，ALU カスケーディングの有無が性能に大きな影響を与える．

以上より，ALU カスケーディングに対しては先行実行することで性能向上率を上げることができ，先行実行に対しては ALU カスケーディングによって Behind thread を効率的に実行できるという関係にあり，相乗効果が期待できる．

5 まとめ

本稿では ALU カスケーディングの評価を行った．その結果，データキャッシュミスによるストールサイクルを減少することで，ALU カスケーディングによる性能向上率が上がることを確認した，さらに，そのための先行実行手法を述べた．

今後，提案した先行実行方式を ALU カスケーディングと組み合わせて実装し，性能評価を行う予定である．

参考文献

[1] 藤枝他．教育・研究に有用な MIPS システムシミュレータ SimMips．情報処理学会論文誌，Vol.50, No.11, pp.2665-2676, November 2009

[2] Chaudhry et al. Simultaneous Speculative Threading:A Novel Pipeline Architecture Implemented in Sun's ROCK Processor. In *Proc. of the 36th Int'l Symp. on Computer Architecture*, pages 484-495, June 2009.

Copyright ©2012 Information Processing Society of Japan.
All Rights Reserved.