社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－ARC－162 （34）
2005－HPC－101 （34）
2005／3／9

# クラスタ型スーパースカラプロセッサにおけるストア命令の早期発行手法

渡邊翔太[†]　入江英嗣[†††,††]
高田正法[††]　坂井修一[††]

クラスタ型アーキテクチャは、実行コアを複数のクラスタに分散し、広い実行幅と高クロック動作の両立を目指している。本研究ではこのようなクラスタ型のスーパースカラプロセッサで従来よりも大きなボトルネックとなってしまうストア命令の In-Order かつ１サイクルに１つという発行の制限に着目し、ストア命令をより早い時刻に発行する手法を提案する。

# Reducing Issue Delay of Store Instructions on A clustered Microarchitecture

Shota Watanabe,[†] Hidetsugu Irie,[†††,††] Masanori Takada[††]
and Shuichi Sakai [††]

Clustered microarchitecture aims at coexistence of wider execution width and higher clock rate by distributing an execution core to clusters. In this paper, we focuse on the issue constraint of store instructions: Store instructions should be issued only one in a cycle following program order. This limit becomes a narrower bottleneck on clustered microarchitecture. Then we propose technique reducing issue delay of store instructions.

## 1. はじめに

将来のより微細なデバイス技術の下では、プロセッサの動作周波数を決める支配的な要素は従来のゲート遅延から配線遅延へと変わっていく。その為、今後はプロセスルールの縮小がそのまま性能の向上には結びつかなくなる事が指摘されている。特に現行のスーパースカラプロセッサにおける、肥大したデータパスは、配線遅延に対して耐性がない[1)]。

その問題に対して、「クラスタ型アーキテクチャ」が次世代のプロセッサアーキテクチャとして注目されている。クラスタ型アーキテクチャは、クラスタと呼ばれる発行幅の小さな実行ユニットを複数持つ構成のアーキテクチャである。１サイクルの長さを決定するタイミングクリティカルパスは、実行幅の増加と共に指数関数的に増大するので、実行ユニットあたりの実行幅が小さいクラスタ型アーキテクチャでは動作周波数のさらなる向上が期待できる[2)3)]。反面、異なるクラスタ間での通信には通信遅延がかかる。その為、１クロック当たりの命令処理性能（IPC）は、現行の集中型プロセッサに比べ低下する。よって、クラスタ型アーキテクチャがポテンシャルを活かす為にはIPCの低下を最小限に留める事、すなわちクラスタ間での通信を最小限に抑えることが必要である。その為には命令をどのクラスタに割り当てるかを決定するステアリングロジックが重要になってくる。特に、クラスタ化の恩恵を受けないキャッシュメモリへのアクセスが相対的に遅くなる為、メモリ操作命令のステアリングの性能は全体の IPC へ大きな影響を及ぼす。

ストア命令はクラスタ型スーパースカラプロセッサの IPC が従来の集中型スーパースカラプロセッサに比べて伸び悩んでいる理由の１つである。それは、通常のスーパースカラプロセッサは、ストア命令を In-Order かつ１サイクルに１つだけ発行する、という設計を用いる事が多いからである。クラスタ型スーパースカラプロセッサにこの設計を適用した場合、従来以上に並列度の抽出を妨げてしまう。なぜなら、クラスタ間通信遅延がかかることと、実行コアに比べてメモリのアクセスレイテンシが相対的に大きくなることが原因である。その為、IPC が低下し動作周波数の面でのメリットを打ち消してしまうことになる。本研究では、ストア命令のステアリング手法の変更と発行条件を緩和することで、このボトルネックを解消し、IPC の向上を目指す。

以下、本論文の構成は次の用になっている。第２章で対象とするアーキテクチャの詳細を述べ、第３章で既存手法の問題点を述べる。第４章で提案する手法を

† 東京大学 工学部
Faculty of Engineering, The University of Tokyo
†† 東京大学大学院 情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
††† 科学技術振興機構
Japan Science and Technology Agency

解説し、その手法を第５章で評価・考察する。第６章で関連する研究について述べ、第７章でまとめを述べる。

## 2. 対象とするアーキテクチャ

### 2.1 ベースラインモデル

図１に対象とするアーキテクチャのブロック図を示す。想定するベースラインモデルは、１命令実行幅のクラスタ８個がネットワークで接続された構成となっており、各クラスタには発行キュー（図中の IQ），複製されたレジスタファイル、演算器、フォーワーディングパスが備わっている。クラスタ間ネットワークのトポロジは今回のモデルでは理想化し、どのクラスタ間の通信にも一律の通信遅延がかかる。また、通信の調停もなく、同時に複数のデータを受信・送信できる。

各命令はフロントエンド処理の後、ステアリングロジックにより、どのクラスタで実行するかが決定され、いずれかのクラスタの IQ へステアリングされる。ステアリングロジックには、Parcerisa らのアルゴリズム[9] と同様、依存と負荷バランスを考慮した戦略を用いる。通常、命令はオペランドが生成されるクラスタにステアリングされ、最高負荷のクラスタと、最低負荷のクラスタの負荷の差が閾値を超えると、最低負荷のクラスタに強制的にステアリングされる。負荷の指標としては、各 IQ 内の wake up 済みだが select されていない命令数を利用する。

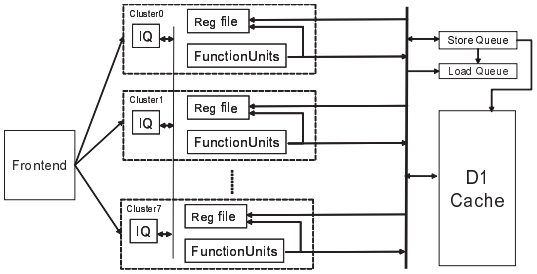


**図 1** 対象とするアーキテクチャ

ベースラインモデルの各パラメータを表 1 に示す。メモリ依存予測手法には、16k entry の Wait Table を用い、100k 命令ごとに Wait Table をリフレッシュする。また、ストア命令は In-Order に発行され、必ず 1 つ前のストア命令が発行するまで自分は発行しない。そのため、ストア命令の実行順序が逆転し、メモリの値が無効化されてしまう Write After Write(WAW) ハザードは発生しない。

Wait Table の予測ミスにより、ロード命令が、不正に親ストア命令を追い越してしまった場合、Read After Write(RAW) ハザードが発生する。ベースラインモデルでは、RAW ハザードが発生した場合、該当ロード命令および、それ以後の命令の実行結果を破棄し、再びロード命令からフェッチし直す。これをパイプライン・フラッシュと呼ぶ。パイプライン・フラッシュが発生すると、先行するすべてのストア命令を待って発行した場合よりも、大きなペナルティになるので、的確な発行条件の設定が重要となる。ベースラインモデルでは、D1 キャッシュは集中型を想定し、全クラスタで共有する。そのため、どのクラスタにメモリ操作命令がステアリングされたとしても、一律のアクセスレイテンシがかかる。

**表 1** ベースラインモデルの各パラメータ

| | |
|---|---|
| クラスタ数 | 8 |
| 各クラスタ内の資源 | 64 entry IQ, 256 物理レジスタ<br>1 命令/cycle |
| フェッチ／リタイア幅 | 最大 16 命令 |
| 命令キャッシュヒット率 | 100% |
| 分岐予測 | gshare predictor 16k entry<br>10bit history |
| メモリ依存予測 | 16k entry Wait Table<br>(100k cycle refresh interval) |
| D1 キャッシュ | 64kB, 2-way SA, 64 byte line<br>8cycle hit latency<br>3 read port, 1 read/write port |
| パイプライン段数 | 16 |
| フロントエンドレイテンシ | 11 cycle |
| Issue to Issue レイテンシ | 1 cycle |
| 整数演算レイテンシ | 1 cycle |
| 整数乗算レイテンシ | 15 cycle |
| 浮動小数点演算レイテンシ | 4 cycle |
| クラスタ間通信遅延 | 2 cycle |

### 2.2 評価環境

プロセッサの性能は “動作周波数× IPC”で与えられるが、最大動作周波数の評価は回路レベルのシミュレーションが必要なため、一般的に評価が難しい。そこで、本研究では評価にトレースベースシミュレータを用い、表 1 に示した構成のクラスタ型スーパースカラプロセッサをベースラインモデルとして、IPC を計測し、性能の指標として用いた。提案する要素技術の効果はベースラインモデルに適宜追加して測定し、特に明記しない部分は表 1 の設定に従った。

シミュレーションモデルの命令セットは DEC Alpha 21264 に準拠し、実行トレースを入力とする。評価には SPEC95int(train) から、compress, gcc, go, ijpeg, li, m88ksim, perl(jumble, primes, scrrable), vortex の 10 種類を用い、先頭から最大 256M 命令について計測した。

## 3. 既存手法の問題点

ストア命令は WAW ハザードを防ぐ為に、現行の Out-of-Order 発行のスーパースカラプロセッサにおいても In-Order に発行される。つまり、各ストア命令

間の In-Order 性を保証するために、フロントエンド処理で自分の１つ前のストア命令の発行を待ってから wake up するように wake up 条件が設定される。この方法をクラスタ型スーパースカラプロセッサに直接適用する場合、クラスタ間では通信遅延が発生するために、従来よりも大きなペナルティになってしまう。連続するストア命令列がそれぞれ異なるクラスタにステアリングされた場合、wake up 条件である自分の１つ前のストア命令の発行を知るのは、実際に発行された時刻のクラスタ間通信遅延時間後になる。その為、多数の連続したストア命令がそれぞれ異なるクラスタにステアリングされた場合、クラスタ間通信遅延のオーバーヘッドが大きくなる。この影響はストア命令だけでなく、ストア命令に依存関係のあるロード命令や、その後ろの命令全体に及び、後続命令の発行を遅らせてしまう。その結果として IPC が低下する。

D1 キャッシュのアクセスポート数を増やす事は、さらなるキャッシュアクセスレイテンシの増加を引き起こすため難しい。また、前述の通りストア命令は自分の１つ前のストア命令が発行されて初めて自分が wake up できる。これらの理由により、ストア命令は１サイクルに１つしか実行できない。ストア命令は全命令の１割程度しかないものの、一般にバースト的に集中して存在することが知られている。そのような場合は、1 [store/cycle] ではストア命令の発行が間に合わない。IQ 内に未発行のストア命令が溜まってしまい、ストア命令に依存のある後続命令が wake up できなくなってしまう。

負荷分散を考慮したステアリング戦略を取ると、多くのストア命令が連続してフェッチされた時それぞれが異なるクラスタにステアリングされる可能性が高まり、クラスタ間通信遅延が何段にも渡ってかかる事となる。これが後続命令の発行を遅らせ、全体として大きなボトルネックになっている。そこで、より高速にストア命令を発行できる手法の必要性が高まっている。

## 4. ストア命令の早期発行手法

### 4.1 Static Store Steering

本研究では、ストア命令を高速に発行するために、「ストア命令専用のクラスタを作り、ストア命令をすべて同じクラスタにステアリングする」というステアリング戦略を提案する。この手法では、ストア命令がすべて同じクラスタ内にあり、１つ前のストア命令の発行を直ちに知ることができるので、発行条件解決の為にクラスタ間通信にかかる時間を待つ必要がない。その反面、一般の命令とは確実に異なるクラスタにステアリングされるので、オペランドの到着には必ず通信遅延がかかってしまう。

図 2 にこの手法のブロック図を示す。また、この提案手法を以後、Static Store Steering と呼ぶことにする。Static Store Steering においては、ストア命令専用クラスタ内には IQ、レジスタ、アドレス演算器のみがあれば良い。通常のクラスタと比べてその他の演算器類を持たない分、占有面積が小さくなりコスト的にも有利になる。

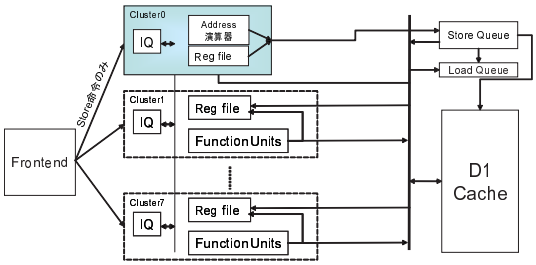


**図 2** 提案するステアリング戦略の概略

### 4.2 Modulo 2 store scheme

4.1 節において、ストア命令を１つのストア専用クラスタに集中させ、クラスタ間通信遅延の削減により高速化をめざす Static Store Steering を提案した。しかし、依然ストア命令は In-Order に発行されるので、最大でも 1 サイクルに 1 つのストア命令しか発行することができなかった。ストア命令は全命令の１割程度しかないが、一般的に集中して存在している。特に、メモリのコピーなどを行うルーチンでは、数百〜数千のストア命令が連続してフェッチされることもある。

ここではさらに、ストア命令専用のクラスタを複数個おくことによって、２つのストア命令を１サイクルに発行できるようなハードウェア構成を提案する。これにより、Static Store Steering 以上の頻度でストア命令を発行することができ、より高い IPC を達成することができる。しかし、WAW ハザードを回避する為、D1 キャッシュへは In-Order に値を書き込むので、キャッシュへは１サイクルに１つの値しか書くことができないので、ストア専用クラスタだけではなく、StoreQueue も複数個配置する必要がある。さらにストア命令の wake up 条件を緩和することにより、１サイクルに複数のストア命令を発行し、後続の命令の早期発行を促すことができる。

図 3 に提案手法の構成を示す。この構成ではストア命令専用のクラスタを２つ作り、フロントエンドでストア命令だけを交互にステアリングする。その上でストア命令の wake up 条件を２つ前のストア命令、つまり同じクラスタにステアリングされた最後のストア命令の発行とする。これにより、同じクラスタにステアリングされたストア命令間では、In-Order に発行される。逆に、他のクラスタのストア命令とは同期を取っていないので、全体としてストア命令は部分的に Out-of-Order に発行されていることになる。クラスタ内でアドレスを計算されたストア命令は、各ストア専用クラスタに対応した StoreQueue に格納され、後

段の Selector で１つずつ交互に D1 キャッシュに書き込むようにする。この段階でキャッシュには In-Order な書き込みが保証され、WAW ハザードを回避することができる。この手法では、２つのストア専用クラスタでそれぞれ独立にストア命令が発行できるので、同時に２つのストア命令を発行することができる。また、自分の２つ前のストア命令は自分と同じストア専用クラスタにステアリングされているので、Static Store Steering と同様に各ストア専用クラスタ内で、通信遅延をかけずに直ちに発行できる。この提案手法を、以後 Modulo 2 store scheme と呼ぶ。

この手法では、Out-of-Order 発行を部分的にとどめることで、完全なストア Out-of-Order 発行をする場合に必要になる複雑な StoreQueue が不必要である。一方、ストア命令の発行順序が In-Order である保証がなくなるので、非投機ロード命令が wake up 条件を、先行する “最後の” ストア命令の発行とした場合でも、先行する “すべての” ストア命令の発行を待った事にならない。その為、非投機ロード命令も RAW ハザードを起こしてしまう危険性がある。RAW ハザードを回避する為に、両方のストア専用クラスタの最後のストア命令の発行を待つように wake up 条件を定めれば解決できるが、２つの依存関係を wake up 条件に設定する為には、IQ にさらなるハードウェアの追加が必要になる為、好ましい手段ではない。

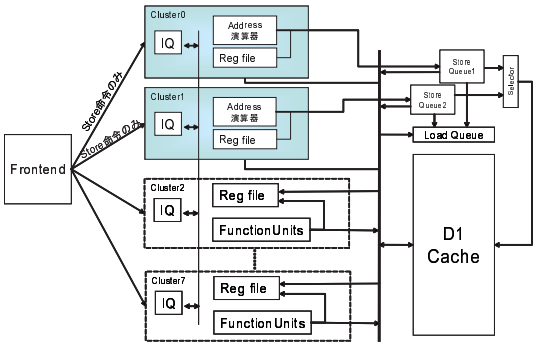


**図 3**　2store/cycle を実現するハードウェア構成

## 5. 提案手法の評価

### 5.1 Static Store Steering の評価

4.1 節に示した Static Store Steering をベースラインモデルに実装し、IPC を評価した。結果は図 4 に示す。 Static Store Steering を適用した場合、Baseline よりも全ベンチマークの調和平均（グラフの HM）で 10.33%の IPC 向上が見られた。これは、主にバースト的にフェッチされたストア命令列が同じクラスタにステアリングされた結果、wake up 遅延が減少した効果が大きいからである。

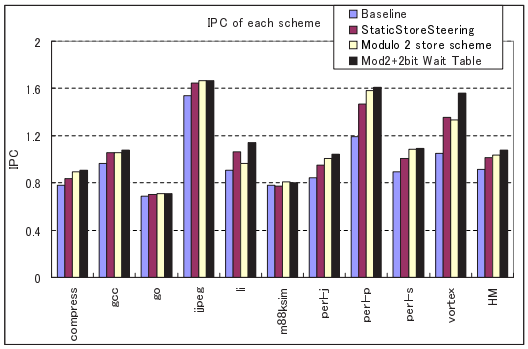


**図 4**　提案手法適用後の IPC

アプリケーション毎に、IPC 向上（m88ksim では若干低下）の度合いが異なるが、これは、wake up 遅延の減少とオペランド到着にかかるクラスタ間通信遅延とのバランスによるものである。つまり、自分の１つ前のストア命令から時間をおいてストア命令が単独でフェッチされた場合、最も重要な発行条件はオペランドの到着であり、ストア命令はオペランドが生成されるクラスタにステアリングされた場合が１番早く発行できる。しかし、連続して複数のストア命令がバースト的にフェッチされた場合、オペランドの到着よりも、自分の１つ前のストア命令が発行したという情報が到着する時刻の方が後になる可能性が高い。後者の場合は Static Store Steering を適用した方が高速に処理できる。その為、アプリケーション毎の IPC 向上の度合いは、プログラム中のストア命令の分布の偏り具合に相関がある。

### 5.2 Modulo 2 store scheme の評価

同様に、Modulo 2 store scheme についても、IPC を計測し、結果は図 4 に併記した。Modulo 2 store scheme の場合、Baseline よりも 12.85%IPC が向上したものの、これは Static Store Steering だけを適用した場合に比べて、2.28%の向上でしかない。これは、図 5 に示すとおり、RAW ハザードが大幅に増加しているのが原因である。ただし、図 5 の縦軸は、「RAW ハザード発生数／実行したロード命令数」を用いた。

4.2 節で述べたように、Modulo 2 store scheme 適用後は非投機ロード命令が確実に RAW ハザード回避できる保証がなくなる。これにより RAW ハザード発生数が大きく増加し、パイプライン・フラッシュによるペナルティが加算され、2[store/cycle] での高速発行のメリットを打ち消してしまっている。

### 5.3 Modulo 2 store scheme における メモリ依存予測の正確性向上

メモリ依存予測手法に Wait Table を用いる場合、Modulo 2 store scheme を適用すると非投機ロード命令が確実に先行するすべてのストア命令の発行を待って発行することができなくなる。その為、RAW ハザー

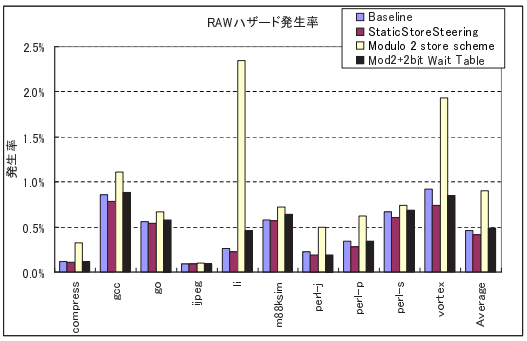


**図 5** 提案手法適用後の RAW ハザードの発生率

ドの発生数が大幅に多くなり、2 [store/cycle] での発行のメリットを打ち消してしまう。それを回避する為に、Wait Table 依存予測を拡張し、Modulo 2 store scheme 適用時に特化したメモリ依存予測方法を提案する。１つの PC に対応する Wait Table のエントリを、従来の 1bit から 2bit へ拡張する。これにより、「依存なし」「１つ前のストア命令に依存あり」「２つ前のストア命令に依存あり」の３つの状態を保持することができる。「１つ前のストア命令に依存あり」と予測されたロード命令が RAW ハザードを起こした場合、次回から「２つ前のストア命令に依存あり」と予測する。FrontEnd でロード命令をデコードする際、PC に対応した Wait Table のエントリを参照し、依存のあると予測されたストア命令の発行を wakeup 条件を設定する。

これにより従来の Wait Table 予測よりも、学習能力を上げる事ができる。反面、各エントリを 2bit にすることによって、同じエントリ数を確保しようとした場合、Wait Table の容量は従来の倍になってしまう。以後この手法を「2bit Wait Table」と呼ぶ事にする。

### 5.4 2bit Wait Table による RAW ハザード低減の評価

同様に図 4 と図 5 に 2bit Wait Table 適用後の Modulo 2 store scheme における IPC と RAW ハザード発生数を併記した。2bit Wait Table を適用することにより、Modulo 2 store scheme の弱点であった RAW ハザードの増加を十分軽減でき、ベースラインモデルから IPC を 17.2%向上させることができた。RAW ハザード発生数も、ベースラインモデルから 4.1%増に留めることができ、通常の 1bit Wait Table 使用時の半分以下とすることができた。

また、各エントリを 2bit とする事により問題になる Wait Table の容量についての考察は図 6 に示す。各測定点は SPEC95 int の 10 種類のベンチマークでの IPC の調和平均（Harmonic Mean）での値である。図 6 より、同じ容量の Wait Table ならば 1Entry=1bit で構成するよりも 1Entry=2bit で構成し、エントリ数を半減させてでもより適切なメモリ依存予測のできる 2bit Wait Table を適用した方が IPC が向上することが判った。

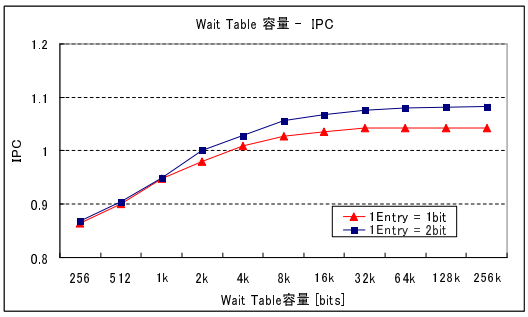


**図 6** Wait Table 容量 - IPC

### 5.5 Modulo 2 store scheme における StoreQueue のオーバーフロー防止手法

Modulo 2 store scheme においては、２つのストア専用クラスタが独立に毎サイクル１つのストア命令を発行できるので、全体では１サイクルに２つのストア命令を発行することができる。しかし、D1 キャッシュへは毎サイクル１つの値しか書き込めない為、バースト的なストア命令列を実行しようとした場合、StoreQueue のエントリがオーバーフローしてしまう恐れがある。それを回避する為に、StoreQueue へ Push されるストア命令を制限することにより、StoreQueue のオーバーフローを防止する手法を提案する。その為には、StoreQueue の残りエントリ数がある閾値よりも少なくなった場合に、ストア命令を Select しなければよい。この手法は、２つのストア専用クラスタにおいてそれぞれ独立に行えばよく、片方の StoreQueue がオーバーフローしそうになったからといって、もう片方のストア専用クラスタが Select を止める必要はない。閾値の設定は、残り StoreQueue エントリ数が「Select されてから StoreQueue に Push されるまでの間にかかるサイクル数」個あればオーバーフローすることはないので、本研究のベースラインモデルにおいては、「最大 StoreQueue エントリ数-3」と設定した。

### 5.6 必要な StoreQueue 長の評価

ベースラインモデルに Modulo 2 store scheme を適用し、5.5 節で述べた StoreQueue のオーバーフロー防止手法を用いて、StoreQueue のエントリ数を制限した場合の IPC を測定した。結果は図 7 に示す。メモリ依存予測手法には 2bit Wait Table を用いた。Modulo 2 store scheme での StoreQueue は２つに分かれており、それぞれのストア専用クラスタと対応付けられている為、１つの Queue あたりの容量は横軸の値の半分になる。また、各測定点は SPEC95 int の 10 種類のベンチマークでの IPC の調和平均での値である。Modulo2 store scheme では、図 7 より、32 エントリ（16 エントリ× 2 個）以上の容量の StoreQueue があ

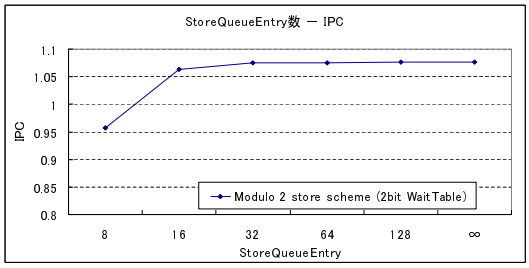


**図 7** StoreQueueEntry － IPC

れば、無限エントリでの IPC とほぼ変わらない結果となった。

## 6. 関 連 研 究

Palacharla らは、演算器のフォーワーディングパスや、発行キューについて回路レベルの解析を行い、デバイス技術が進むほど、実行幅によるタイミング・クリティカルパスの増加が大きくなることを示し、クラスタ型構成の有効性を議論した[3]。現状でも DEC の Alpha 21264 がクラスタ型プロセッサに該当する[4]。スケジューラもクラスタ化されるようなより積極的なモデルについて、Canal ら[5] はリネーム時に動的に命令実行クラスタを決定するステアリングロジックを追加することで、従来とバイナリ互換を持つクラスタ型アーキテクチャを提案した。クラスタ型アーキテクチャが性能を発揮する為にはステアリングロジックの最適化が重要である。Percerisa ら[9] は命令間のレジスタ依存とクラスタ負荷の双方をバランスを考慮するステアリング手法を提案した。

## 7. お わ り に

本論文では、クラスタ型スーパースカラプロセッサにおいて、従来よりも問題となるストア命令の In-Order かつ 1 サイクルに 1 つという制限に焦点を当てて議論を行った。ストア命令の wake up 条件は、オペランドの到着よりも、前のストア命令の発行の方がよりクリティカルになる事が多いという仮定の下、ストア命令を強制的に同じクラスタにステアリングするという Static Store Steering を提案した。それにより、ベースラインモデルよりも IPC を 10.33%向上させることができた。

また、ストア命令の発行条件を緩和することで、1 サイクルに 2 つのストア命令を発行できる Moduro 2 store Scheme を提案した。これにより、ストア命令の発行は高速になったものの、非投機ロード命令の wake up 条件が曖昧になり、RAW ハザードの発生数が増加した。その為、パイプライン・フラッシュによるペナルティが大きくなり、IPC はベースラインモデルから 12.85%向上するにとどまった。また、ストア命令を 1 サイクルに 2 つ発行しても、キャッシュへは 1 サイクルに 1 つしか書き込めない為、StoreQueue がオーバーフローしてしまうという問題もあった。これらの Modulo2store scheme の問題点への対策として、Wait Table の各エントリを 2bit に拡張する 2bit Wait Table と、StoerQueue の空きエントリ数によって Select を止めるという SelectLogic の改変を提案した。この 2 つの対策を適用する事により、RAW ハザードの低減と StoreQueue のオーバーフローの防止を同時に実現でき、最終的にベースラインモデルから 17.22%の IPC 向上を達成できた。

**謝辞** 本論文の研究は，一部 21 世紀 COE 「情報技術戦略コア」、及び科学技術振興機構 CREST「ディペンダブル情報基盤」による。

## 参 考 文 献

1) M.T.Bohr,“Interconnect Scaling - The Real Limiter to High Performance ULSI”, In 1995 IEEE International Electron Devices Meeting Technical Digest, pp.241-224, 1995
2) 入江英嗣 “クラスタ化プロセッサにおける分散投機メモリフォーワーディング手法の研究”, 情報処理学会論文誌 コンピューティングシステム Vol.45 No.SIG7, Oct, 2004
3) S.Palacharla, N.P.Jouppi, J.E.Smith, “Complexity-Effective Superscalar Processors”, 24th Int.Symp.on Computer Architecture, pp.1-13, Jun 1997.
4) R.Kessler, “The Alpha 21264 Microprocessor”, IEEE Micro, vol.19, no.2, pp.24-36, Mar/Apr 1999.
5) R.Canal, J.M.Parcerisa, and A.Gonzalez, “A Cost-Effective Clustered Architecture”, Int.Conf.on Parallel Architectures and Compilation Techniques, pp.160-168, 1999.
6) J.M.Parcerisa, A.Gonzalez, “Reducing Wire Delay Penalty through Value Prediction”, 33rd Int.Symp.on Microarchitecture, pp.317-326, Dec 2000.
7) 服部直也 “発行時間差に基づいた命令ステアリング方式”, 情報処理学会論文誌 コンピューティングシステム Vol.45 No.SIG7, Oct, 2004
8) G.Z.Chrysos, J.S.Emer, “Memory Dependence Prediction using Store Sets”, 25th Int.Symp.on Compuetr Architecture, pp.142-153, Jul 1998.
9) J.M.Parcerisa, J.Sahuquillo, A.Gonzalez, J.Duato, “Efficient Interconnects for Clustered Microarchitectures”, In Proc. Of the Int. Conf. on Parallel Architectures and Compilation Techniques(PACT2002), pp.291-390