2007－ARC－174 (2)
2007／8／1

ツインテール・アーキテクチャの改良

亘 理 靖 展[†] 堀 尾 一 生[†††] 入 江 英 嗣[††]
五 島 正 裕[†] 坂 井 修 一[†]

本研究室で提案しているツインテール・アーキテクチャでは，発行幅を増やさずにスーパスカラ・プロセッサに演算器を追加することで実質的に発行幅が増えたような効果が得られる．ツインテール・アーキテクチャでは並列にメモリ・アクセス可能なロード命令が増えることで大きな性能向上が得られる．しかし，プロセッサ内のロード命令の数を増やすためにはロード・ストアキューのサイズを大きくする必要があり，配線遅延の増大を招く可能性がある．本論文では，ロード・ストアキューからアクセス・オーダ・バイオレーションの検出機構を分離し，アクセス・オーダ・バイオレーションの検出をするバッファを別途設けることで，ツインテール・アーキテクチャにおいて，配線遅延の増大を招くことなく，同時にメモリ・アクセスできるロード命令を増加させるモデルを提案する．シミュレーションによる提案モデルの評価では，ツインテール・アーキテクチャにおいてアクセス・オーダ・バイオレーション検出時の再実行方法を理想的にしたモデルとほぼ同等の IPC の向上が得られた．

Improvement of Twintail Architecture

Yasuhiro Watari,[†] Kazuo Horio,[†††] Hidetsugu Irie,[††]
Masahiro Goshima[†] and Shuichi Sakai[†]

We propose Twintail Architecture, an architecture which gives effect similar to widening issue width but does not lead to greater latency. Twintail Architecture contributes to superscalar processor's throughput by enabling paralell memory access. However, it seems to provoke wiring delay with enlarging the size of load/store queue for the purpose of increasing in-flight load instructions. In this paper, we propose an reasonable model which increases the number of in-flight load instructions, by decoupling the function of access order violation detection from the load/store queue and enlarging a buffer which detects access order violation. Evaluation showed proposed model improves IPC as well as ideal re-execution model.

1. はじめに

スーパスカラ・プロセッサに演算器を追加して同時実行可能な命令数を増やすためには発行幅を増やす必要がある．研究によると，命令ウィンドウのセル面積は発行幅の 3 乗に比例する.[3] 面積が増えるとその平方根に比例して配線長が長くなるので，発行幅の増加は配線長の増大につながる．

微細化されたプロセスで製造された LSI においてはゲート遅延より配線遅延が支配的である．従ってスーパスカラ・プロセッサの発行幅を増やすと，命令ウィンドウ内の遅延が増大し，今後クロック速度の向上の足枷となる可能性がある．

† 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
††† シャープ株式会社
SHARP Corporation

本研究室では発行幅を増やさずにスーパスカラ・プロセッサに演算器を追加して命令を実行し，実質的に発行幅が増えたような効果を得ることができるアーキテクチャ，ツインテール・アーキテクチャを提案している.[1][2]

ツインテール・アーキテクチャでは，実質的に発行幅が増加する効果だけでなく，実質的に命令ウィンドウのエントリ数が増加する効果や，分岐予測ミス・ペナルティが軽減される効果など，様々な効果によってスーパスカラ・プロセッサのスループットを向上させる．

特にツインテール・アーキテクチャにおいて，実質的に命令ウィンドウのエントリ数が増加する効果により，並列してメモリ・アクセスをするロード命令の数が増えることが，大きくパフォーマンスを向上させる．

本論文ではツインテール・アーキテクチャにおいて，並列にメモリ・アクセス可能なロード命令を増やすことができる現実的なモデルの提案をし，シミュレーションによる評価を行う．

本論文の構成は以下の通りである．2 章でツインテール・アーキテクチャの構成を説明する．3 章でツイン

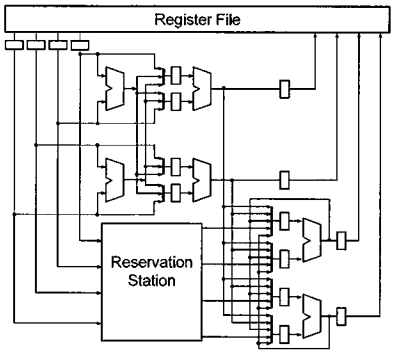

図 1　ツインテール・アーキテクチャのブロック図
Fig. 1　Block diagram of Twintail Architecture

テール・アーキテクチャによる性能向上の原理を説明する．4 章でツインテール・アーキテクチャの改良モデルについて述べる．5 章にはシミュレーションによる評価結果を掲げる．6 章で本研究の結論と今後の課題を述べる．

2. ツインテール・アーキテクチャの構成

本章ではツインテール・アーキテクチャの構成について説明する．

2.1 ディスパッチ前に実行できる命令

ツインテール・アーキテクチャはリザベーション・ステーション/ リオーダ・バッファを用いるスーパスカラ・プロセッサをベースとして想定している．リザベーション・ステーション/ リオーダ・バッファを用いるスーパスカラ・プロセッサではレジスタ読み出しはディスパッチ前に行う．レジスタ読み出し後，全ての命令はリザベーション・ステーションへ書き込まれ，ウェイクアップ，セレクト，発行のステップを経て実行される．レジスタ読み出し直後にオペランドが揃っている命令はスケジューリング・ロジックを経ずに実行可能であるが，スーパスカラ・プロセッサではこのような命令もリザベーション・ステーションへ書き込み，スケジューリング・ロジックを経て実行する．

ツインテール・アーキテクチャはレジスタ読み出し直後にオペランドが揃った命令をスケジューリング・ロジックを経ずに実行するため，専用の演算器をレジスタ・ファイルの直下に配置する．

2.2 ツインテール・アーキテクチャの演算器の配置

ツインテール・アーキテクチャには，図 1 のようにレジスタ・ファイルの直下の演算器と命令ウィンドウから発行された命令を実行する演算器の 2 つの実行系が存在する．ツインテール・アーキテクチャではレジスタ・ファイル直下の演算器からなる実行系を**インオーダーテール**，命令ウィンドウとそこから発行された命令を実行する演算器からなる実行系を**アウトオブオーダーテール**と呼ぶ．

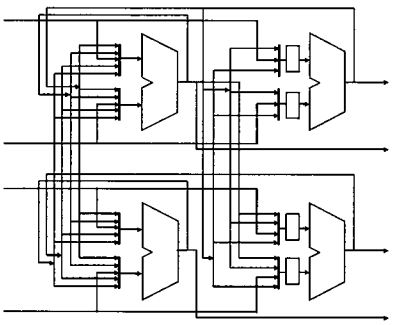

図 2　インオーダーテールのブロック図
Fig. 2　Block diagram of In-order Tail

レジスタ読み出し後，命令ウィンドウ (アウトオブオーダーテール) へのディスパッチとインオーダーテールでの実行は並列に行われる．インオーダーテールで実行される命令はスケジューリング・ロジックを経ずに実行されるので，実行までのレイテンシは大幅に削減される．

本章では以降，インオーダーテールの構成について述べる．

2.3 インオーダーテール

2.3.1 インオーダーテールの演算器の配置

インオーダーテールにはフェッチ幅と同数の演算器を多段に並べて配置する．図 2 にフェッチ幅 2 × 2 段の演算器を配置する場合のブロック図を示す．

スーパスカラ・プロセッサにおいて同時にフェッチされた命令群を**フェッチ・グループ**と呼ぶ．インオーダーテールは命令をフェッチ・グループ単位で取り込み，各フェッチ・グループを毎サイクル次段の演算器に移動する．

1 段目の演算器ではレジスタ読み出し直後にオペランドが揃った命令を実行する。インオーダーテール内で実行された命令の結果を後続の命令が利用可能にするため，インオーダーテールでは後段の演算器から前段の演算器に実行結果をバイパスする．アウトオブオーダーテールにもインオーダーテールで実行された命令の結果をバイパスする．インオーダーテール内の全ての演算器からアウトオブオーダーテールへのバイパスを行うとバイパス・ネットワークが複雑になるため，アウトオブオーダーテールへのバイパスはインオーダーテールの最終段の演算器から行う．アウトオブオーダーテールからインオーダーテールへはオペランドのバイパスを行わない．

2.3.2 インオーダーテールで実行する命令

インオーダーテールでは整数演算命令，分岐命令，ロード命令を実行する．それぞれの命令のインオーダーテール内での実行可能性は，ソース・オペランドがインオーダーテール内で揃うかどうかによって判断可能である，インオーダーテール内での実行可能性の

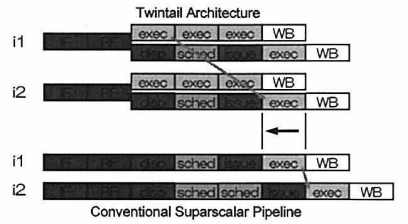

図 3 アウトオブオーダーテールの命令が即座に発行される
Fig. 3 Consumer instruction at Out-of-order tail is issued immediately

判断にはソース・オペランドのレディネスの情報が必要になるが，エントリの中身までは知る必要がない．そのため，インオーダーテールでの実行可能性の判断はレジスタ読み出しと並行して行える．

インオーダーテールで実行可能と判断された命令はアウトオブオーダーテールへのディスパッチを止める．こうすることで，インオーダーテールで命令が実行されると，アウトオブオーダーテールでの発行幅や命令ウィンドウのエントリ数に余裕が生まれる．

3. ツインテール・アーキテクチャによる性能向上

ツインテール・アーキテクチャは以下のような効果によってスーパスカラ・プロセッサのスループットを向上させる．

- 分岐予測ミス・ペナルティが軽減される効果
- 実質的に発行幅が増加する効果
- インオーダーテールで実行された命令のレイテンシが隠蔽される効果
- 実質的に命令ウィンドウのエントリ数が増加する効果

本章ではここに挙げた 4 つの効果について説明する．

3.1 分岐予測ミス・ペナルティが軽減される効果

インオーダーテールで分岐命令の結果が得られると，分岐予測ミスの発覚が早くなる．従って分岐予測ミス・ペナルティが軽減される．

3.2 インオーダーテールで実行された命令のレイテンシが隠蔽される効果

図 3 の上はツインテール・アーキテクチャのパイプライン，下は通常のスーパスカラ・プロセッサのパイプラインである．IF はフェッチ，RF はレジスタ読み出し，disp はディスパッチ，sched はスケジューリング，issue は発行，exec は実行，WB は書き戻しの，それぞれパイプライン・ステージを表わす．図に示したのは命令 i2 が命令 i1 に依存するプログラムの実行の様子である．通常のスーパスカラ・プロセッサでは，命令 i1 の実行終了と同時にその結果を利用できるように，命令 i2 は 1 サイクル遅れてウェイクアップされる．ツインテール・アーキテクチャでは，命令 i1 が

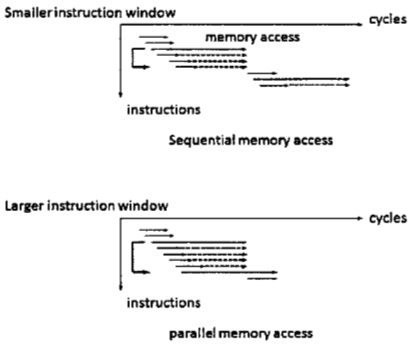

図 4 命令ウィンドウの大きさがメモリ・アクセスの並列性を制限する
Fig. 4 Size of instruction window limits parallel access to memory

インオーダーテールで実行された場合には早期に結果が得られるため，命令 i2 は命令ウィンドウへ入り次第発行できる．

3.3 実質的に発行幅/命令ウィンドウのエントリ数が増加する効果

インオーダーテールで実行可能と判明した命令はアウトオブオーダーテールへのディスパッチを行わない．従ってツインテール・アーキテクチャではインオーダーテールで実行される命令の分，アウトオブオーダーテールの発行幅と命令ウィンドウのエントリ数に余裕を生むことができる．

キャッシュ・ミスが連続するようなプログラムのパフォーマンスは，メモリ・アクセスが並列に行われるかシーケンシャルに行われるかに大きく支配される．そのようなプログラムでは，インオーダーテールで実行された命令の分，命令ウィンドウのエントリ数に余裕が生まれることで，並列してメモリ・アクセスを行えるロード命令の数が増え，パフォーマンスが大きく改善されることがわかった[1]．命令ウィンドウの大きさがメモリ・アクセスの並列性を制限する仕組みを図 4 に示す．

4. ツインテール・アーキテクチャの改良

3.3 節において述べたように，並列にメモリ・アクセスを行えるロード命令の数を増やせれば，パフォーマンスの改善が期待できる．しかしながらプロセッサ内のロード・ストア命令の数を増やすため，ロード・ストアキューのエントリ数を増やすとロード・ストアキューのスケジューリング・ロジックが複雑になり，配線遅延が増大する恐れがある．このため，単純にロード・ストアキューのエントリ数を増加させて並列にメモリ・アクセスを行えるロード命令の数を増やすことは望ましくない．

本章ではツインテール・アーキテクチャにおいて現

実的な構成で，並列にメモリ・アクセスを行えるロード命令の数を増やすことができるモデルの提案をする．まず，スーパスカラ・プロセッサのロード・ストアキューについて述べる．

4.1 ロード・ストアキュー

スーパスカラ・プロセッサではロード・ストア命令は命令ウィンドウへのディスパッチ時にインオーダにロード・ストアキューに入る．ロード・ストアキューに入った命令は，アドレス計算が終了した命令から順にアウトオブオーダにメモリ・ユニットに発行され，メモリ・アクセスを開始する．メモリ・アクセスが完了したロード・ストア命令はロード・ストアキューの先頭から順に，アクセス・オーダ・バイオレーションの検出をされたのち，インオーダにリタイアを行う．

ディスパッチされた全てのロード・ストア命令はロード・ストアキュー内に存在し，インフライトなロード命令を増やすためには，ロード・ストアキューの大きさを広げる必要がある．しかしながら，メモリ・ユニットに発行するロード・ストア命令を選択するスケジューリング・ロジックの配線遅延は，ロード・ストアキューの大きさに比例して増えるため，ロード・ストアキューを大きくすることでメモリ・アクセスを並列に行えるロード・ストア命令を増やすことは望ましくない．

ツインテール・アーキテクチャでは，インオーダーテールで実行されるロード命令はアクセス・オーダ・バイオレーションの検出を行う必要はあるが，ロード・ストアキューのスケジューリング・ロジックを経てメモリ・ユニットに発行が行われる必要はない．そこで，ロード・ストアキューの機能を分離し，アウトオブオーダーテールにおいてメモリ・ユニットへロード・ストア命令を発行するためのキューと，インオーダーテールおよびアウトオブオーダーテールで実行されたロード・ストア命令のアクセス・オーダ・バイオレーションを検出するためのバッファに分ける．以下，アウトオブオーダーテールでメモリ・ユニットへのロード・ストア命令の発行を行うキューをロード・ストアキュー，ロード・ストア命令のアクセス・オーダ・バイオレーションを検出するバッファをアクセス・オーダ・バイオレーション・ディテクション・バッファ(AOVD バッファ) と呼ぶ．

アクセス・オーダ・バイオレーションの検出は，ロード・ストア命令のメモリ・アクセス時に，AOVD バッファ内に存在する自分より後続の命令の中に，同じアドレスに既にアクセスしている命令が存在するかを調べることで行う．命令のリタイアは AOVD バッファの先頭の命令から順に行えばよいので 複雑なスケジューリング・ロジックは必要ない．そのため AOVD バッファのエントリ数はロード・ストアキューのエントリ数に比べ容易に増加させやすい．

次節でロード・ストアキューおよび AOVD バッファを用いたツインテール・アーキテクチャの改良モデルについて述べる．

4.2 ツインテール・アーキテクチャの改良モデル

インオーダーテールで実行可能と判断したロード命令はインオーダーテールへ命令を送る時に AOVD バッファへ挿入する．インオーダーテールで実行されるロード命令はインオーダーテールの最終段からメモリ・アクセスを開始するため，ロード・ストアキューには挿入しない．インオーダーテールで実行不可能なロード・ストア命令はアウトオブオーダーテールへのディスパッチ時に，ロード・ストアキューおよび AOVD バッファへ挿入する．

メモリ・アクセスを行っているロード・ストア命令はすべて AOVD バッファ内に存在するので，AOVD バッファのエントリ数を増やすことで，並列してメモリ・アクセスを行うロード命令の数を増やすことができる．

アクセス・オーダ・バイオレーションの検出時の再実行の方式としては，アクセス・オーダ・バイオレーションを起こした命令に依存する命令を選択的に再実行する方式と，アクセス・オーダ・バイオレーションを起こした命令より後続の命令をすべてフェッチからやり直す方式が考えられる．

選択的再実行方式では，再実行時にアクセス・オーダ・バイオレーションを起こした命令に依存する命令のみをレジスタ読み出しから実行し直す．しかしながら，インオーダーテール内で実行されている命令は命令ウィンドウ内に存在していないため，そのような命令を選択的に再実行することは難しい．

再フェッチしてやり直す方式では，インオーダーテール内で実行される命令の再実行も容易であるが，後続の命令全てをフェッチして再実行するため，アクセス・オーダ・バイオレーションが頻発すると IPC の低下が起きてしまう．この影響を低減するため，高性能なアドレス一致・不一致予測器であるストア・セット[4]を用い，アクセス・オーダ・バイオレーションの発生を抑える．ストア・セットによって先行のストアにアドレスが一致すると予測されたロード・ストア命令は，アウトオブオーダーテールへディスパッチし，先行のストアに対してインオーダに実行する．

次節ではアクセス・オーダ・バイオレーションの検出時に再フェッチしてやり直す方式のシミュレーションを行い，評価をする．

5. 評　　価

シミュレーションにより本論文で提案するツインテール・アーキテクチャのモデルについて性能評価を行った．

5.1 評価環境

シミュレーションには本研究室で開発したシミュレータ「鬼斬」を用いた．CINT2000，CFP2000 のプロ

グラムの最初の 1G 命令をスキップし，100M 命令を実行した．表 1 にベース・モデルの主要なパラメータを示す．

5.2 ツインテール・アーキテクチャの構成

評価対象のツインテール・アーキテクチャは以下のような構成を想定している．

- インオーダーテールの演算器は 4 ウェイ× 3 段
- インオーダーテールでは整数命令，分岐命令，ロード命令を実行する

5.3 IPC

ベースのスーパスカラ・プロセッサはアクセス・オーダ・バイオレーションの検出時に，選択的再実行をする．ツインテール・アーキテクチャのモデルとしては，アクセス・オーダ・バイオレーションが検出された際，再フェッチからやり直すモデルと，インオーダーテール内の命令も選択的に再実行可能とした理想的なモデルを評価した．評価を行ったツインテール・アーキテクチャのモデルを表 2 に示す．

ベースとなるスーパスカラ・プロセッサと，ツインテール・アーキテクチャの評価モデルで IPC の比較を行った結果を，図 5 に示す．グラフはベース・モデルの IPC を 1 として正規化してある．また全モデルの IPC の平均向上率を表 3 に示す．

AOVD バッファのエントリ数を大きくすることにより，フェッチして再実行を行うモデルでも，選択的再実行を行う理想的なモデルとほぼ同等の IPC 向上率が得られた．これは，ストア・セット予測を利用したことによりアクセス・オーダ・バイオレーションの発生がほとんど抑えられ，フラッシュ再実行のペナルティが低く抑えられていることや，AOVD バッファのエントリ数がロード・ストアキューの大きさの 2 倍ほどあれば，フェッチされるほぼすべてのロード命令を AOVD バッファ内に蓄えておくことが可能なためである．

179.art や 189.lucas では，AOVD バッファの大きさを大きくすると，パフォーマンスが大幅に改善される．これは 3.3 節で述べたように，AOVD バッファを大きくすることで，並列にメモリ・アクセスできるロード命令が増加する効果による．逆に 171.swim や 188.ammp において性能向上率がすべてのモデルで同程度になっているのは，インフライトなロード命令が少なく，AOVD バッファの大きさによって性能が制限されないためである．

またインオーダーテールで実行する種類の命令が，性能向上が大きかった 179.art の AOVD バッファを 32 エントリにしたモデルにおいて，それぞれインオーダーテール・アウトオブオーダーテールのどこで実行されたのかを図 6 に示す．iBC は整数の条件分岐命令，iLD は整数のロード命令，iALU は整数演算命令，fLD は浮動小数点のロード命令，ADDR はアドレス計算命令を表している．図 6 では，インオーダーテールで実行する種類の命令はその大部分がインオーダーテー

表 1 ベース・モデルの主要なパラメータ
Table 1 Principal parameters of the base model

Fetch Width	4
Issue Width	4
Integer Units	ALU × 4, MUL × 2, DIV × 2, 16entries instruction queue(used universally by all integer instructions)
Floating Point Units	ADD/MUL × 2, DIV × 1, 16entries instruction queue(used universally by all FP instructions)
Memory Instructions Queue	16entries
L1 Cache	16KB, 4ways, 3cycles to access
L2 Cache	1MB, 4ways, 15cycles to access
Memory	200cycles to access
Physical Registers	128
Re-order Buffer Entries	128
Pipeline Depth	Fetch-3, Read-2, Dispatch-2, Schedule-1, Issue-2, Write Back-2
Branch Prediction	BTB: 2K entries, gshare: 32K entries PHT, 10 bits branch history
Address Match Prediction	Store Set: 4k entries Store Set ID table, 512 entries Last Fetched Store Table

表 2 評価対象のモデル
Table 2 Models evaluated

twintail-s	ツインテール・アーキテクチャで選択的再実行を行う．AOVD バッファのエントリ数：無限
twintail-16	ツインテール・アーキテクチャでフラッシュ再実行を行う．AOVD バッファのエントリ数：16
twintail-24	ツインテール・アーキテクチャでフラッシュ再実行を行う．AOVD バッファのエントリ数：24
twintail-32	ツインテール・アーキテクチャでフラッシュ再実行を行う．AOVD バッファのエントリ数：32

表 3 IPC の平均向上率
Table 3 Average IPC improvement

twintail-s	18.7%
twintail-16	10.6%
twintail-24	16.9%
twintail-32	18.2%

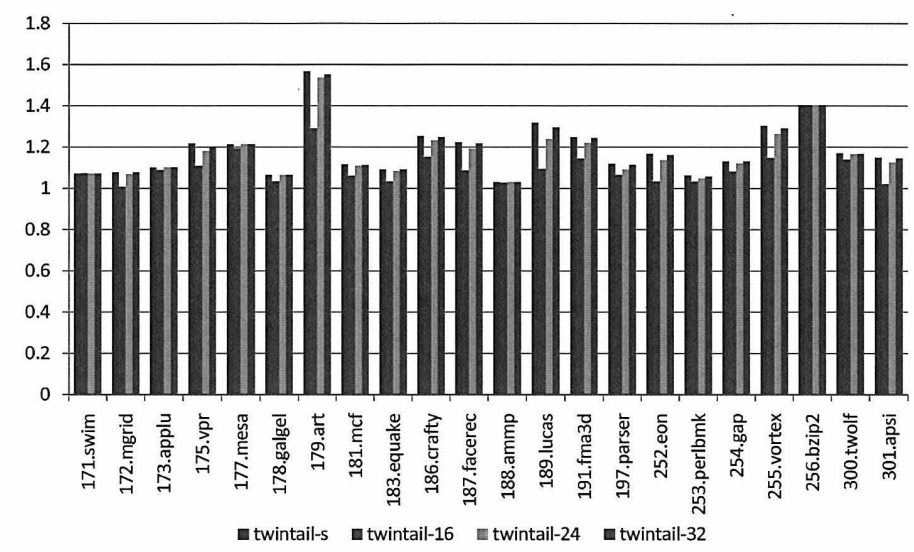

図 5 ツインテール・アーキテクチャの IPC
Fig. 5 IPC of Twintail Architecture

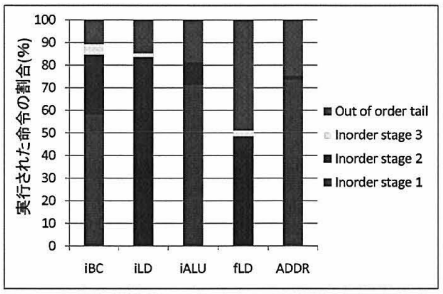

図 6 179.art の命令実行の様子

ル内で実行されている．ロード命令はアドレス計算が終了した後実行可能になるので，インオーダーテールの 2 段目以降から開始される．

6. まとめと今後の課題

ツインテール・アーキテクチャでは発行幅を増やさずにスーパスカラ・プロセッサに演算器を追加してそこで命令を実行し，実質的に発行幅が増えたような効果を得ることができる．特に並列してメモリ・アクセスを行えるロード命令の数が増えることによる性能向上が大きく，本論文ではスーパスカラ・プロセッサのロード・ストアキューからアクセス・オーダ・バイオレーションの機構を分離させることで，このようなロード命令の数をツインテール・アーキテクチャによって増加させることができる現実的なモデルを提案した．

シミュレーションによる評価では AOVD バッファのエントリ数を大きくすることにより，選択的再実行を行った理想的モデルと同等の性能を得ることができることがわかった．

今後の課題としては，詳細なツインテール・アーキテクチャの評価を行い，インオーダーテール内でのストア命令の実行や，インオーダーテールで実行される命令の割合を積極的に増加させる改良を行うことが挙げられる．

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処理基盤」による．

参 考 文 献

1) 堀尾 一生，平井 遥，五島 正裕，坂井 修一: ツインテール・アーキテクチャ, 先進的計算基盤システムシンポジウム *SACSIS2007*, pp.303-311, May, 2007
2) 平井遥, 入江英嗣，五島正裕，坂井修一: ツインテール・アーキテクチャ, *2006-ARC-169*, pp.43-48, July, 2006
3) 五島正裕: *Out-of-order ILP* プロセッサにおける命令スケジューリングの高速化の研究., 京都大学 *(*博士論文*)*, March, 2004
4) Chrysos, G.Z.; Emer, J.S.: *Memory dependence prediction using store sets*, *The 25th Annual International Symposium on Computer Architecture*, pp.142-153, July, 1998